Secure Outsourced Garbled Circuit Evaluation for Mobile Devices

Henry Carter, Georgia Institute of Technology
Benjamin Mood, University of Oregon
Patrick Traynor, Georgia Institute of Technology
Kevin Butler, University of Oregon

A

Garbled circuits provide a powerful tool for jointly evaluating functions while preserving the privacy of each user's inputs. While recent research has made the use of this primitive more practical, such solutions generally assume that participants are symmetrically provisioned with massive computing resources. In reality, most people on the planet only have access to the comparatively sparse computational resources associated with their mobile phones, and those willing and able to pay for access to public cloud computing infrastructure cannot be assured that their data will remain unexposed. We address this problem by creating a new SFE protocol that allows mobile devices to securely outsource the majority of computation required to evaluate a garbled circuit. Our protocol, which builds on the most efficient garbled circuit evaluation techniques, includes a new outsourced oblivious transfer primitive that requires significantly less bandwidth and computation than standard OT primitives and outsourced input validation techniques that force the cloud to prove that it is executing all protocols correctly. After showing that our extensions are secure in the malicious model, we conduct an extensive performance evaluation for a number of standard SFE test applications as well as a privacy-preserving navigation application designed specifically for the mobile use-case. Our system reduces execution time by 98.92% and bandwidth by 99.95% for the edit distance problem of size 128 compared to non-outsourced evaluation. These results show that even the least capable devices are capable of evaluating some of the largest garbled circuits generated for any platform.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Algorithms, Performance, Security

Additional Key Words and Phrases: Garbled Circuits, Mobile Privacy, Secure Function Evaluation

1. INTRODUCTION

Secure Function Evaluation (SFE) allows two parties to compute the result of a function without either side having to expose their potentially sensitive inputs to the other. While considered a generally theoretical curiosity even after the discovery of Yao's garbled circuit [Yao 1982], recent advances in this space have made such computation increasingly practical. Today, functions as complex as AES-128 and approaching one billion gates in size are possible at reasonable throughputs, even in the presence of a malicious adversary.

While recent research has made the constructions in this space appreciably more performant, the majority of related work makes a crucial assumption - *that both parties are symmetrically provisioned with massive computing resources.* For instance, Kreuter et al. [Kreuter et al. 2012] rely on the Ranger cluster at the Texas Advanced Computing Center to compute their results using 512 cores. In reality, the extent of a user's computing power may be their mobile phone, which has many orders of magnitude less computational ability. Moreover, even with access to a public compute cloud such as Amazon EC2 or Windows Azure, the sensitive nature of the user's data and the history of data leakage from cloud services [Thomas 2010; Rash 2012] prevent the direct porting of known SFE techniques.

In this paper, we develop mechanisms for the secure outsourcing of SFE computation from constrained devices to more capable infrastructure. Our protocol maintains the privacy of both participant's inputs and outputs while significantly reducing the computation and network overhead required by the mobile device for garbled circuit evaluation. We develop a number of extensions to allow the mobile device to check for malicious behavior from the circuit generator or the cloud and a novel Outsourced Oblivious Transfer for sending garbled input data to the cloud. We then implement the new protocol on a commodity mobile device and reasonably provisioned servers and demonstrate significant performance improvements over evaluating garbled circuits directly on the mobile device.

We make the following contributions:

— **Outsourced oblivious transfer & outsourced consistency checks:** Instead of blindly trusting the cloud with sensitive inputs, we develop a highly efficient Outsourced Oblivious Transfer primitive that allows mobile devices to securely delegate the majority of computation associated with oblivious transfers. We also provide mechanisms to outsource consistency checks to prevent a malicious circuit generator from providing corrupt garbled values. These checks are designed in such a way that the computational load is almost exclusively on the cloud, but cannot be forged by a malicious or "lazy" cloud. We demonstrate that both of our additions are secure in the malicious model as defined by Kamara et al. [Kamara et al. 2012].

— **Performance Analysis:** Extending upon the implementation by Kreuter et al. [Kreuter et al. 2012], we conduct an extensive performance analysis against a number of simple applications (e.g., edit distance) and cryptographic benchmarks (e.g., AES-128). Our results show that outsourcing SFE provides improvements to both execution time and bandwidth overhead. For the edit distance problem of size 128, we reduce execution time by 98.92% and bandwidth by 99.95% compared to direct execution without outsourcing on the mobile device.

— **Privacy Preserving Navigation App:** To demonstrate the practical need for our techniques, we design and implement an outsourced version of Dijkstra's shortest path algorithm as part of a Navigation mobile app. Our app

provides directions for a Presidential motorcade without exposing its location, destination, or known hazards that should be avoided (but remain secret should the mobile device be compromised). *The optimized circuits generated for this app represent the largest circuits evaluated to date.* Without our outsourcing techniques, such an application is far too processor, memory and bandwidth intensive for any mobile phone.

While this work is similar in function and provides equivalent security guarantees to the Salus protocols recently developed by Kamara et al. [Kamara et al. 2012], our approach is dramatically different. The Salus protocol framework builds their scheme on a completely different assumption, specifically, that they are outsourcing work from low-computation devices with *high communication bandwidth*. With provider-imposed bandwidth caps and relatively slow and unreliable cellular data connections, this is not a realistic assumption when developing solutions in the mobile environment. Moreover, rather than providing a proof-of-concept work demonstrating that offloading computation is possible, this work seeks to develop and thoroughly demonstrate the practical potential for evaluating large garbled circuits in a resource-constrained mobile environment.

The remainder of this work is organized as follows: Section 2 presents important related work and discusses how this paper differs from Salus; Section 3 provides cryptographic assumptions and definitions; Section 4 formally describes our protocols; Section 5 provides informal security discussion; Section 6 provides full proofs of security; Section 7 shows the results of our extensive performance analysis; Section 8 presents our privacy preserving navigation application for mobile phones; and Section 9 provides concluding remarks.

2. RELATED WORK

Three different general techniques have been developed to perform secure function evaluation. The first, using homomorphic encryption, has yielded several special-purpose protocols using partially homomorphic encryption [Osadchy et al. 2010; Carter et al. 2013; Bendlin et al. 2011; Damgard et al. 2012]. Other more generalizable protocols combine partially homomorphic encryption with garbled circuit constructions [Henecka et al. 2010]. However, these schemes are manually optimized for specific functions. While protocols using fully homomorphic encryption promise more generalizable results [Gentry et al. 2012], these cryptosystems are currently too inefficient to be used practically, even on server-class machines. A second technique for secure function evaluation uses only the Oblivious Transfer (OT) primitive to compute certain functions [Naor and Pinkas 1999]. Due to the high costs generally associated with OT primitives, these techniques have received less attention, as they also generally perform less efficiently. The most studied and widely applied technique is the garbled circuit, first proposed by Yao [Yao 1982]. With the Fairplay implementation [Malkhi et al. 2004], garbled circuits were shown to have significant promise in terms of efficiency.

Beginning with Fairplay [Malkhi et al. 2004], several secure two-party computation implementations and applications have been developed using Yao garbled circuits [Yao 1982] in the semi-honest adversarial model [Brickell and Shmatikov 2005; Kruger et al. 2006; Iliev and Smith 2010; Huang et al. 2011; Jha et al. 2008; Lindell and Pinkas 2000; Malka 2011; Nipane et al. 2011]. However, a malicious party using corrupted inputs or circuits can learn more information about the other party's inputs in these constructions [Kiraz and Schoenmakers 2006]. To resolve these issues, new protocols have been developed to achieve security in the malicious model, using cut-and-choose constructions [Lindell and Pinkas 2011], input commitments [shelat and Shen 2011], and other various techniques [Mohassel and Franklin 2006; Kiraz 2008]. To improve the performance of these schemes in both the malicious and semi-honest adversarial models, a number of circuit optimization techniques have also been developed to reduce the cost of generating and evaluating circuits [Kolesnikov and Schneider 2008; Choi et al. 2012; Goyal et al. 2008; Mood et al. 2012]. Kreuter et al. [Kreuter et al. 2012; Kreuter et al. 2013] combined several of these techniques into a general garbled circuit protocol that is secure in the malicious model and can efficiently evaluate circuits on the order of billions of gates using parallelized server-class machines. This SFE protocol is currently the most efficient implementation that is fully secure in the malicious model. (The dual execution construction by Huang et al. leaks one bit of input [Huang et al. 2012].)

Garbled circuit protocols rely on oblivious transfer schemes to exchange certain private values. While several OT schemes of various efficiencies have been developed [Bellare and Micali 1990; Naor and Pinkas 2001; Peikert et al. 2008; Lindell and Pinkas 2011], Ishai et al. demonstrated that any of these schemes can be extended to reduce k^c oblivious transfers to k oblivious transfers for any given constant c [Ishai et al. 2003]. Using this extension, exchanging potentially large inputs to garbled circuits became much less costly in terms of cryptographic operations and network overhead. Even with this drastic improvement in efficiency, oblivious transfers still tend to be a costly step in evaluating garbled circuits.

Currently, the performance of garbled circuit protocols executed directly on mobile devices has been shown to be feasible only for small circuits in the semi-honest adversarial model [Carter et al. 2013; Huang et al. 2011]. While outsourcing general computation to the cloud has been widely considered for improving the efficiency of applications running on mobile devices, the concept has yet to be widely applied to cryptographic constructions. Green et al. began exploring this idea by outsourcing the costly decryption of ABE ciphertexts to server-class machines while still maintaining data privacy [Green et al. 2011]. Considering the costs of exchanging inputs and evaluating garbled circuits securely, an outsourcing technique would be useful in allowing limited capability devices to execute SFE protocols.

Naor et al. [Naor et al. 1999] develop an oblivious transfer technique that sends the chooser's private selections to a third party, termed a proxy. While this idea is applied to a limited application in their work, it could be leveraged more generally into existing garbled circuit protocols. Our work develops a novel extension to this technique to construct a garbled circuit evaluation protocol that securely outsources computation to the cloud.

In work performed concurrently and independently from our technique, Kamara et al. recently developed two protocols for outsourcing secure multiparty computation to the cloud in their Salus system [Kamara et al. 2012]. While their work achieves similar functionality to ours, we distinguish our work in the following ways: first, their protocol is constructed with the assumption that they are outsourcing work from devices with low-computation but high-bandwidth capabilities. With cellular providers imposing bandwidth caps on customers and cellular data networks providing highly limited data transmission speed, we construct our protocol without this assumption using completely different cryptographic constructions. Second, their work focuses on demonstrating outsourced SFE as a proof-of-concept. Our work offers a rigorous performance analysis on mobile devices, and outlines a practical application that allows a mobile device to participate in the evaluation of garbled circuits that are orders of magnitude larger than those evaluated in the Salus system. Finally, their protocol that is secure in the malicious model requires that all parties share a secret key, which must be generated in a secure fashion before the protocol can be executed. Our protocol does not require any shared information prior to running the protocol, reducing the overhead of performing a multiparty fair coin tossing protocol a priori. While our work currently considers only the two-party model, by not requiring a preliminary multiparty fair coin toss, expanding our protocol to more parties will not incur the same expense as scaling such a protocol to a large number of participants. To properly compare security guarantees, we apply their security definitions in our analysis.

3. ASSUMPTIONS AND DEFINITIONS

To construct a secure scheme for outsourcing garbled circuit evaluation, some new assumptions must be considered in addition to the standard security measures taken in a two-party secure computation. In this section, we discuss the intuition and practicality of assuming a non-colluding cloud, and we outline our extensions on standard techniques for preventing malicious behavior when evaluating garbled circuits. Finally, we conclude the section with formal definitions of security.

3.1. Non-collusion with the cloud

Throughout our protocol, we assume that none of the parties involved will ever collude with the cloud. This requirement is based in theoretical bounds on the efficiency of garbled circuit evaluation and represents a realistic adversarial model. The fact that theoretical limitations exist when considering collusion in secure multiparty computation has been known and studied for many years [Chaum et al. 1988; Ben-Or et al. 1988; Lindell 2008], and other schemes considering secure computation with multiple parties require similar restrictions on who and how many parties may collude while preserving security [Canetti et al. 2002; Damgård and Ishai 2006; Damgård and Nielsen 2007; Kamara et al. 2012; Kamara et al. 2011]. Kamara et al. [Kamara et al. 2012] observe that if an outsourcing protocol is secure when both the party generating the circuit and the cloud evaluating the circuit are malicious and colluding, this implies a secure two-party scheme where one party has sub-linear work with respect to the size of the circuit, which is currently only possible with fully homomorphic encryption. However, making the assumption that the cloud will not collude with the participating parties makes outsourcing securely a theoretical possibility. In reality, many cloud providers such as Amazon or Microsoft would not allow outside parties to control or affect computation within their cloud system for reasons of trust and to preserve a professional reputation. In spite of this assumption, we cannot assume the cloud will always be semi-honest. For example, our protocol requires a number of consistency checks to be performed by the cloud that ensure the participants are not behaving maliciously. Without mechanisms to force the cloud to make these checks, a "lazy" cloud provider could save resources by simply returning that all checks verified without actually performing them. Thus, our adversarial model encompasses a non-colluding but potentially malicious cloud provider that is hosting the outsourced computation.

3.2. Attacks in the malicious setting

When running garbled circuit based secure multiparty computation in the malicious model, a number of well- documented attacks exist. We address here how our system counters each.

Malicious circuit generation: In the original Yao garbled circuit construction, a malicious generator can garble a circuit to evaluate a function f' that is not the function f agreed upon by both parties and could compromise the security of the evaluator's input. To counter this, we employ an extension of the random seed technique developed by Goyal et al. [Goyal et al. 2008] and implemented by Kreuter et al. [Kreuter et al. 2012]. Essentially, the technique uses a cut-and-choose, where the generator commits to a set of circuits that all presumably compute the same function. The parties then use a fair coin toss to select some of the circuits to be evaluated and some that will be re-generated and hashed by the cloud given the random seeds used to generate them initially. The evaluating party then inspects

the circuit commitments and compares them to the hash of the regenerated circuits to verify that all the check circuits were generated properly.

Selective failure attack: If, when the generator is sending the evaluator's garbled inputs during the oblivious transfer, he lets the evaluator choose between a valid garbled input bit and a corrupted garbled input, the evaluator's ability to complete the circuit evaluation will reveal to the generator which input bit was used. To prevent this attack, we use the input encoding technique from Lindell and Pinkas [Lindell and Pinkas 2007], which lets the evaluator encode her input in such a way that a selective failure of the circuit reveals nothing about the actual input value. To prevent the generator from swapping garbled wire values, we use a commitment technique employed by Kreuter et al. [Kreuter et al. 2012].

Input consistency: Since multiple circuits are evaluated to ensure that a majority of circuits are correct, it is possible for either party to input different inputs to different evaluation circuits, which could reveal information about the other party's inputs. To keep the evaluator's inputs consistent, we again use the technique from Lindell and Pinkas [Lindell and Pinkas 2007], which sends all garbled inputs for every evaluation circuit in one oblivious transfer execution. To keep the generator's inputs consistent, we use the malleable claw-free collection construction of shelat and Shen [shelat and Shen 2011]. This technique is described in further detail in Section 4.

Output consistency: When evaluating a two-output function, we ensure that outputs of both parties are kept private from the cloud using an extension of the technique developed by Kiraz [Kiraz and Schoenmakers 2006]. The outputs of both parties are XORed with random strings within the garbled circuit, and the cloud uses a witness-indistinguishable zero-knowledge proof as in the implementation by Kreuter et al. [Kreuter et al. 2012]. This allows the cloud to choose a majority output value without learning either party's output or undetectably tampering with the output. At the same time, the witness-indistinguishable proofs prevent Alice and Bob from learning the index of the majority circuit. This prevents Bob from learning anything by knowing which circuit evaluated to the majority output value.

3.3. Malleable claw-free collections

To prevent the generating party from providing different inputs for each evaluation circuit, we implement the malleable claw-free collections technique developed by shelat and Shen [shelat and Shen 2011]. Their construction essentially allows the generating party to prove that all of the garbled input values were generated by exactly one function in a function pair, while the ability to find an element that is generated by both functions implies that the generator can find a claw.

In their work, they provide the following definitions.

DEFINITION 1. *Claw-free Collections [shelat and Shen 2011; Goldreich and Kahan 1995]: A three-tuple of algorithms (G, D, F) is a claw-free collection if the following conditions hold.*

(*1*) **Easy to evaluate:** *Both the index selecting algorithm G and the domain sampling algorithm D are probabilistic polynomial-time, while the evaluating algorithm F is a deterministic polynomial-time.*
(*2*) **Identical range distribution:** *Let $f_I^b(x)$ be the output of F given input (b, I, x). For any $I \in range(G)$, the random variable $f_I^0(D(0, I))$ and $f_I^1(D(1, I))$ are identically distributed.*
(*3*) **Hard to form claws:** *For every non-uniform probabilistic polynomial-time algorithm A, every polynomial $p(\cdot)$, and every sufficiently large n, it is true that $Pr[I \leftarrow G(1^n); (x, y) \leftarrow A(I) : f_I^0(x) = f_I^1(y)] < \frac{1}{p(n)}$.*

Given this definition, they build their scheme on a modified construction which adds in a malleability property. This allows the consistency of the inputs to be checked with a witness-indistinguishable proof.

DEFINITION 2. *Malleable Claw-Free Collection [shelat and Shen 2011]: A four-tuple of algorithms (G, D, F, R) is a malleable claw-free collection if the following conditions hold:*

(*1*) **A subset of claw-fee collections:** *(G, D, F) is a claw-free collection, and the range of D and F are groups, denoted by $(\mathbb{G}_1, \star)$ and $(\mathbb{G}_2, \diamond)$ respectively.*
(*2*) **Uniform domain sampling:** *For any I in the range of G, random variable $D(0, I)$ and $D(1, I)$ are uniform over $\mathbb{G}_1$, and denoted by $D(I)$ for simplicity.*
(*3*) **Malleability:** *$R : \mathbb{G}_1 \rightarrow \mathbb{G}_2$ runs in polynomial time, and for $b \in \{0, 1\}$, any I in the range of G, and any $m_1, m_2 \in \mathbb{G}_1, f_I^b(m_1 \star m_2) = f_I^b(m_1) \diamond R_I(m_2)$.*

Our implementation of a malleable claw-free collection uses the same construction as Kreuter et al. [Kreuter et al. 2012], built on the discrete logarithm assumption.

3.4. Model and Definitions

The work of Kamara et al. [Kamara et al. 2012] presents a definition of security based on the ideal-model/real-model security definitions common in secure multiparty computation. Because their definition formalizes the idea of a non-colluding cloud, we apply their definitions to our protocol for the two-party case in particular. We summarize their definitions below.

Real-model execution. The protocol takes place between two parties (P_1, P_2) executing the protocol and a server P_3, where each of the executing parties provides input x_i, auxiliary input z_i, and random coins r_i and the server provides only auxiliary input z_3 and random coins r_3. In the execution, there exists some subset of independent parties $(A_1, ..A_m), m \leq 3$ that are malicious adversaries. Each adversary corrupts one executing party and does not share information with other adversaries. For all honest parties, let OUT_i be its output, and for corrupted parties let OUT_i be its view of the protocol execution. The i^{th} partial output of a real execution is defined as:

$$REAL^{(i)}(k, x; r) = \{OUT_j : j \in H\} \cup OUT_i$$

where H is the set of honest parties and r is all random coins of all players.
Ideal-model execution. In the ideal model, the setup of participants is the same except that all parties are interacting with a trusted party that evaluates the function. All parties provide inputs x_i, auxiliary input z_i, and random coins r_i. If a party is semi-honest, it provides its actual inputs to the trusted party, while if the party is malicious or non-colluding, it provides arbitrary input values. In the case of the server P_3, this means simply providing its auxiliary input and random coins, as no input is provided to the function being evaluated. Once the function is evaluated by the trusted third party, it returns the result to the parties P_1 and P_2, while the server P_3 does not receive the output. If a party aborts early or sends no input, the trusted party immediately aborts. For all honest parties, let OUT_i be its output to the trusted party, and for corrupted parties let OUT_i be some value output by P_i. The ith partial output of an ideal execution in the presence of some set of independent simulators is defined as:

$$IDEAL^{(i)}(k, x; r) = \{OUT_j : j \in H\} \cup OUT_i$$

where H is the set of honest parties and r is all random coins of all players. In this model, the formal definition of security is as follows:

DEFINITION 3. *A protocol securely computes a function f if there exists a set of probabilistic polynomial-time (PPT) simulators $\{Sim_i\}_{i\in[3]}$ such that for all PPT adversaries $(A_1, \cdots, A_3)$, x, z, and for all $i \in [3]$:*

$$\{REAL^{(i)}(k, x; r)\}_{k\in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k\in N}$$

Where $S = (S_1, \cdots, S_3)$, $S_i = Sim_i(A_i)$, and r is random and uniform.

4. PROTOCOL

Our protocol can be divided into five phases, illustrated in Figure 1. Given a circuit generator Bob, and an evaluating mobile device Alice, the protocol can be summarized as follows:

— Phase 1: Bob generates a number of garbled circuits, some of which will be checked, others will be evaluated. After Bob commits to the circuits, Alice and Bob use a fair coin toss protocol to select which circuits will be checked or evaluated. For the check circuits, Bob sends the random seeds used to generate the circuits to the Cloud and the hashes of each circuit to Alice. These are checked to ensure that Bob has not constructed a circuit that is corrupted or deviates from the agreed-upon function.
— Phase 2: Alice sends her inputs to Bob via an outsourced oblivious transfer. Bob then sends the corresponding garbled inputs to the Cloud. This allows the Cloud to receive Alice's garbled inputs without Bob or the Cloud ever learning her true inputs.
— Phase 3: Bob sends his garbled inputs to the Cloud, which verifies that they are consistent for each evaluation circuit. This prevents Bob from providing different inputs to different evaluation circuits.
— Phase 4: The Cloud evaluates the circuit given Alice and Bob's garbled inputs. Since the Cloud only sees garbled values during the evaluation of the circuit, it never learns anything about either party's input or output. Since both output values are blinded with one-time pads, they remain private even when the Cloud takes a majority vote.
— Phase 5: The Cloud sends the encrypted output values to Alice and Bob, who are guaranteed its authenticity through the use of commitments and zero-knowledge proofs.

4.1. Participants

Our protocols reference three different entities:
Evaluator: The evaluating party, called Alice, is assumed to be a mobile device that is participating in a secure two-party computation.
Generator: The party generating the garbled circuit, called Bob, is an application- or web- server that is the second party participating with Alice in the secure computation.
Proxy: The proxy, called Cloud, is a third party that is performing heavy computation on behalf of Alice, but is not trusted to know her input or the function output.

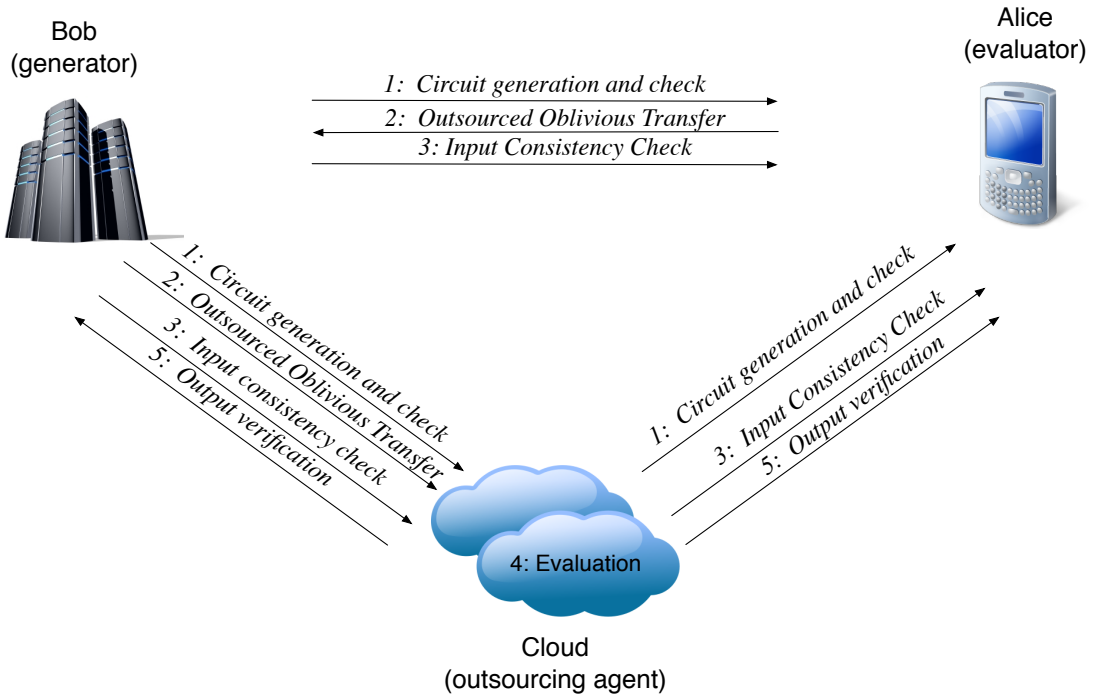


Fig. 1: The complete outsourced SFE protocol.

4.2. Outsourced Protocol

Common inputs: a function $f(x, y)$ that is to be securely computed, a claw-free collection $(G_{CLW}, D_{CLW}, F_{CLW}, R_{CLW})$, two hash functions $H_1 : \{0,1\}^* \rightarrow \{0,1\}^n$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^w$, a primitive 1-out-of-2 oblivious transfer protocol, a perfectly hiding commitment scheme $com_H(key, message)$, and security parameters for the number of circuits built k, the number of primitive oblivious transfers t, and the number of encoding bits for each of Alice's input wires ℓ.
Private inputs: The generating party Bob inputs a bit string b and a random string of bits b_r that is the length of the output string. The evaluating party Alice inputs a bit string a and a random string of bits a_r that is the length of the output string. Assume without loss of generality that all input and output strings are of length $|a| = n$.
Output: The protocol outputs separate private values fa for Alice and fb for Bob.
Phase 1: Circuit generation and checking

(1) *Circuit preparation:* Before beginning the protocol, both parties agree upon a circuit representation of the function $f(a, b)$, where the outputs of the function may be defined separately for Alice and Bob as $f_A(a, b)$ and $f_B(a, b)$. The circuit must also meet the following requirements:
   (a) Additional XOR gates must be added such that Bob's output is set to $fb = f_B(a, b) \oplus b_r$ and Alice's output is set to $fa = f_A(a, b) \oplus a_r$.
   (b) For each of Alice's input bits, the input wire w_i is split into ℓ different input wires $w_{j,i}$ such that $w_i = w_{1,i} \oplus w_{2,i} \oplus \cdots \oplus w_{l,i}$ following the input encoding scheme by Lindell and Pinkas [Lindell and Pinkas 2007]. This prevents Bob from correlating a selective failure attack with any of Alice's input bit values.
(2) *Circuit garbling:* the generating party, Bob, constructs k garbled circuits using a circuit garbling technique $Garble(\cdot, \cdot)$. When given a circuit representation C of a function and random coins rc, $Garble(C, rc)$ outputs a garbled circuit GC that evaluates C. Given the circuit C and random coins $rc_1 \cdots rc_k$, Bob generates garbled circuits $Garble(C, rc_i) = GC_i$ for $i = 1 \cdots k$. For Bob's j^{th} input wire on the i^{th} circuit, Bob associates the value $H_2(\beta_{b,j,i})$ with the input value b, where $\beta_{b,j,i} = F_{CLW}(b, I, \alpha_{b,j,i})$. For Alice's j^{th} input wire, Bob associates the value $H_2(\delta_{b,j,i})$ with the input value b, where $\delta_{b,j,i} = F_{CLW}(b, I, \gamma_{b,j,i})$. All the values $\alpha_{b,j,i}$ and $\gamma_{b,j,i}$ for $b = \{0,1\}, j = 1 \cdots n, i = 1 \cdots k$ are selected randomly from the domain of the claw-free pair using D.
(3) *Circuit commitment:* Bob generates commitments for all circuits by hashing $H_1(GC_i) = HC_i$ for $i = 1 \cdots k$. Bob sends these hashes to Alice. In addition, for every output wire $w_{b,j,i}$ for $b = \{0,1\}$, $j = 1 \cdots n$ and $i = 1 \cdots k$, Bob generates commitments $CO_{j,i} = com_H(ck_{j,i}, (H_2(w_{0,j,i}), H_2(w_{1,j,i})))$ using commitment keys $ck_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots k$ and sends them to both Alice and the Cloud.

Inputs: Alice has a string of encoded input bits ea of length $\ell \cdot n$ and Bob has pairs of input values $(x_{0,j}, x_{1,j})$ for $j = 1 \cdots \ell \cdot n$.

(1) **Setup:** Alice generates random matrix T of size $\ell \cdot n \times t$, Bob generates random string s of length t.
(2) **Primitive OT:** Alice and Bob execute t 1-out-of-2 oblivious transfers with Alice inputting $(T^i, T^i \oplus ea)$ and Bob inputting selection bits s (T^i denotes the i^{th} column of the T matrix). Bob sets the resulting columns as matrix Q.
(3) **Permuting the output:** Alice generates random string p of length $\ell \cdot n$ and sends it to Bob.
(4) **Encrypting the output:** Bob sets the encrypted output pairs $y_{0,j}, y_{1,j}$ where $y_{b,j} = x_{b,j} \oplus H_1(j, Q_j \oplus (b \cdot s))$ (Q_j denotes the j^{th} row of the Q matrix).
(5) **Permuting the outputs:** Bob permutes the encrypted output pairs as $y_{0 \oplus p_j, j}, y_{1 \oplus p_j, j}$ and sends the resulting set of pairs Y to the Cloud.
(6) **Decrypting the output:** Alice sends $h = ea \oplus p$ and T to the Cloud. The Cloud recovers $z_j = y_{h_j, j} \oplus H_1(j, T_j)$ for $j = 1 \cdots \ell \cdot n$ (T_j denotes the j^{th} row of the T matrix).

Fig. 2: The Outsourced Oblivious Transfer protocol

(4) *Input label commitment:* Bob commits to Alice's garbled input values as follows: for each generated circuit $i = 1 \cdots k$ and each of Alice's input wires $j = 1 \cdots \ell \cdot n$, Bob creates a pair of commitment keys $ik_{0,j,i}, ik_{1,j,i}$ and commits to the input wire label seeds $\delta_{0,j,i}$ and $\delta_{1,j,i}$ as $CI_{b,j,i} = com_H(ik_{b,j,i}, \delta_{b,j,i})$. For each of Alice's input wires $j = 1 \cdots \ell \cdot n$, Bob randomly permutes the commitments within the pair $CI_{0,j,i}, CI_{1,j,i}$ across every $i = 1 \cdots k$. This prevents the Cloud from correlating the location of the commitment with Alice's input value during the OOT phase.
(5) *Cut and choose:* Alice and Bob then run a fair coin toss protocol to agree on a set of circuits that will be evaluated, while the remaining circuits will be checked. The coin toss generates a set of indices $Chk \subset \{1, \cdots, k\}$ such that $|Chk| = \frac{3}{5}k$, as in shelat and Shen's cut-and-choose protocol [shelat and Shen 2011]. The remaining indices are placed in the set Evl for evaluation, where $|Evl| = e = \frac{2}{5}k$. For every $i \in Chk$, Bob sends rc_i and the values $[\alpha_{b,1,i}, \cdots, \alpha_{b,n,i}]$ and $[\gamma_{b,1,i}, \cdots, \gamma_{b,\ell \cdot n,i}]$ for $b = \{0, 1\}$ to the Cloud. Bob also sends all commitment keys $ck_{j,i}$ for $j = 1 \cdots n$ and $i \in Chk$ to the Cloud. Finally, Bob sends the commitment keys $ik_{b,j,i}$ for $b = \{0, 1\}$, $i \in Chk$, and $j = 1 \cdots \ell \cdot n$ to the Cloud. The Cloud then generates $Garble(C, rc_i) = GC'_i$ for $i \in Chk$. For each $i \in Chk$, the Cloud then hashes each check circuit $H_1(GC'_i) = HC'_i$ and checks that:
— each commitment $CO_{j,i}$ for $j = 1 \cdots n$ is well formed
— the value $H_2(\beta_{b,j,i})$ is associated with the input value b for Bob's j^{th} input wire
— the value $H_2(\delta_{b,j,i})$ is associated with the input value b for Alice's j^{th} input wire
— for every bit value b and input wire j, the values committed in $CI_{b,j,i}$ are correct
If any of these checks fail, the Cloud immediately aborts. Otherwise, it sends the hash values HC'_i for $i \in Chk$ to Alice. For every $i \in Chk$, Alice checks if $HC_i = HC'_i$. If any of the hash comparisons fail, Alice aborts.

Phase 2: Outsourced Oblivious Transfer (OOT)

(1) *Input encoding:* For every bit $j = 1 \cdots n$ in her input a, Alice sets encoded input ea_j as a random string of length ℓ such that $ea_{1,j} \oplus ea_{2,j} \oplus \cdots \oplus ea_{\ell,j} = a_j$ for each bit in ea_j. This new encoded input string ea is of length $\ell \cdot n$.
(2) *OT setup:* Alice initializes an $\ell \cdot n \times t$ matrix T with uniformly random bit values, while Bob initializes a random bit vector s of length t. See Figure 2 for a more concise view.
(3) *Primitive OT operations:* With Alice as the sender and Bob as the chooser, the parties initiate t 1-out-of-2 oblivious transfers. Alice's input to the i^{th} instance of the OT is the pair $(T^i, T^i \oplus ea)$ where T^i is the i^{th} column of T, while Bob's input is the i^{th} selection bit from the vector s. Bob organizes the t selected columns as a new matrix Q.
(4) *Permuting the selections:* Alice generates a random bit string p of length $\ell \cdot n$, which she sends to Bob.
(5) *Encrypting the commitment keys:* Bob generates a matrix of keys that will open the committed garbled input values and proofs of consistency as follows: for Alice's j^{th} input bit, Bob creates a pair $(x_{0,j}, x_{1,j})$, where $x_{b,j} = [ik_{b,j,Evl_1}, ik_{b,j,Evl_2}, \cdots, ik_{b,j,Evl_e}] || [\gamma_{b_j,j,Evl_2} \star (\gamma_{b_j,j,Evl_1})^{-1}, \gamma_{b_j,j,Evl_3} \star (\gamma_{b_j,j,Evl_1})^{-1}, \cdots, \gamma_{b_j,j,Evl_e} \star (\gamma_{b_j,j,Evl_1})^{-1}]$ and Evl_i denotes the i^{th} index in the set of evaluation circuits. For $j = 1 \cdots \ell \cdot n$, Bob prepares $(y_{0,j}, y_{1,j})$ where $y_{b,j} = x_{b,j} \oplus H_1(j, Q_j \oplus (b \cdot s))$. Here, Q_j denotes the j^{th} row in the Q matrix. Bob permutes the entries using Alice's permutation vector as $(y_{0 \oplus p_j, j}, y_{1 \oplus p_j, j})$. Bob sends this permuted set of ciphertexts Y to the Cloud.
(6) *Receiving Alice's garbled inputs:* Alice blinds her input as $h = ea \oplus p$ and sends h and T to the Cloud. The Cloud recovers the commitment keys and consistency proofs $x_{b,j} = y_{h_j, j} \oplus H_1(j, T_j)$ for $j = 1 \cdots \ell \cdot n$. Here, h_j denotes the j^{th} bit of the string h and T_j denotes the j^{th} row in the T matrix. Since for every $j \in Evl$, the Cloud only has

the commitment key for the b garbled value (not the $b \oplus 1$ garbled value), the Cloud can correctly decommit only the garbled labels corresponding to Alice's input bits.

(7) *Verifying consistency across Alice's inputs:* Given the decommitted values $[\delta_{b,1,i}, \cdots, \delta_{b,\ell \cdot n,i}]$ and the modified pre images $[\gamma_{b_j,j,Evl_2} \star (\gamma_{b_j,j,Evl_1})^{-1}, \gamma_{b_j,j,Evl_3} \star (\gamma_{b_j,j,Evl_1})^{-1}, \cdots, \gamma_{b_j,j,Evl_e} \star (\gamma_{b_j,j,Evl_1})^{-1}]$, the Cloud checks that:

$$\delta_{b_j,j,i} = \delta_{b_j,j,Evl_1} \diamond R_{CLW}(I, \gamma_{b_j,j,i} \star (\gamma_{b_j,j,Evl_1})^{-1})$$

for $i = 2 \cdots e$. If any of these checks fails, the Cloud aborts the protocol.

Phase 3: Generator input consistency check

(1) *Delivering inputs:* Bob delivers the hash seeds for each of his garbled input values $[\beta_{b_1,1,i}, \beta_{b_2,2,i}, \cdots, \beta_{b_n,n,i}]$ for every evaluation circuit $i \in Evl$ to the Cloud, which forwards a copy of these values to Alice. Bob then proves the consistency of his inputs by sending the modified preimages $[\alpha_{b_j,j,Evl_2} \star (\alpha_{b_j,j,Evl_1})^{-1}, \alpha_{b_j,j,Evl_3} \star (\alpha_{b_j,j,Evl_1})^{-1}, \cdots, \alpha_{b_j,j,Evl_e} \star (\alpha_{b_j,j,Evl_1})^{-1}]$ such that $F_{CLW}(b_i, I, \alpha_{b_i,j,i}) = \beta_{b_i,j,i}$ for $j = 1 \cdots n$ and $i \in Evl$ such that GC_i was generated with the claw-free function pair indexed at I.

(2) *Check consistency:* Alice then checks that all the hash seeds were generated by the same function by checking if:

$$\beta_{b_j,j,i} = \beta_{b_j,j,Evl_1} \diamond R_{CLW}(I, \alpha_{b_j,j,i} \star (\alpha_{b_j,j,Evl_1})^{-1})$$

for $i = 2 \cdots e$. If any of these checks fails, Alice aborts the protocol.

Phase 4: Circuit evaluation

(1) *Evaluating the circuit:*For each evaluation circuit, the Cloud evaluates $GC_i(ga_i, gb_i)$ for $i \in Evl$ in the pipelined manner described by Kreuter et al. in [Kreuter et al. 2012]. Each circuit produces two garbled output strings, (gfa_i, gfb_i).

(2) *Checking the evaluation circuits:* Once these output have been computed, the Cloud hashes each evaluation circuit as $H_1(GC_i) = HC_i'$ for $i \in Evl$ and sends these hash values to Alice. Alice checks that for every $i, HC_i = HC_i'$. If any of these checks do not pass, Alice aborts the protocol.

Phase 5: Output check and delivery

(1) *Committing the outputs:*The Cloud then generates random commitment keys ka_i, kb_i and commits the output values to their respective parties according to the commitment scheme defined by Kiraz [Kiraz and Schoenmakers 2006], generating $CA_{j,i} = commit(ka_{j,i}, gfa_{j,i})$ and $CB_{j,i} = commit(kb_{j,i}, gfb_{j,i})$ for $j = 1 \cdots n$ and $i = 1 \cdots e$. The Cloud then sends all CA to Alice and CB to Bob.

(2) *Selection of majority output:* Bob opens the commitments $CO_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots e$ for both Alice and the Cloud. These commitments contain the mappings from the hash of each garbled output wire $H_2(w_{b,j,i})$ to real output values $b_{j,i}$ for $j = 1 \cdots n$ and $i = 1 \cdots e$. The Cloud selects a circuit index maj such that the output of that circuit matches the majority of outputs for both Alice and Bob. That is, $fa_{maj} = fa_i$ and $fb_{maj} = fb_i$ for i in a set of indices IND that is of size $|IND| > \frac{e}{2}$

(3) *Proof of output consistency:* Using the OR-proofs as described by Kiraz [Kiraz and Schoenmakers 2006], the Cloud proves to Bob that CB contains valid garbled output bit values based on the de-committed output values from the previous step. The Cloud then performs the same proof to Alice for her committed values CA. Note that these proofs guarantee the output was generated by one of the circuits, but the value maj remains hidden from both Alice and Bob.

(4) *Output release:* The Cloud then decommits gfa_{maj} to Alice and gfb_{maj} to Bob. Given these garbled outputs and the bit values corresponding to the hash of each output wire, Alice recovers her output string fa, and Bob recovers his output string fb.

(5) *Output decryption:* Alice recovers her output $f_A(a, b) = fa \oplus a_r$, while Bob recovers $f_B(a, b) = fb \oplus b_r$.

5. SECURITY GUARANTEES

In this section, we provide a summary of the security mechanisms used in our protocol and and informal discussion of the security guarantees of our outsourced oblivious transfer construction.

Recall from Section 3 that there are generally four security concerns when evaluating garbled circuits in the malicious setting. To solve the problem of malicious circuit generation, we apply the random seed check variety of cut-&-choose developed by Goyal et al. [Goyal et al. 2008]. To solve the problem of selective failure attacks, we employ the input encoding technique developed by Lindell and Pinkas [Lindell and Pinkas 2007]. To prevent an adversary from using inconsistent inputs across evaluation circuits, we employ the witness-indistinguishable proofs from shelat and Shen [shelat and Shen 2011]. Finally, to ensure the majority output value is selected and not tampered with, we use the XOR-and-prove technique from Kiraz [Kiraz and Schoenmakers 2006] as implemented by Kreuter et al. [Kreuter

et al. 2012]. In combination with the standard semi-honest security guarantees of Yao garbled circuits, these security extensions secure our scheme in the malicious security model.

5.1. Garbled Circuit Generation

To ensure the evaluated circuits are generated honestly, we require two properties. First, we limit the generator Bob's ability to trick Alice into evaluating a corrupted circuit using a cut-&-choose technique similar to a typical, two-party garbled circuit evaluation. Second, we ensure that a lazy Cloud attempting to conserve system resources cannot bypass the circuit checking step without being discovered. We call this the Cloud's "proof-of-work".

CLAIM 1. *Security: Assuming that the hash function $H_1(\cdot)$ is a one-way, collision-resistant hash and that the commitment scheme used is fully binding, then the generator Bob has at best a $2^{-0.32k}$ probability of tricking Alice into evaluating a majority of corrupted circuits, where k is the number of circuits generated.*

shelat and Shen [shelat and Shen 2011] perform a rigorous analysis of the optimal cut-&-choose strategy for evaluating garbled circuits in the malicious setting. Given that the generator prepares k circuits before the cut-&-choose, their protocol sets the number of check circuits to $\frac{3k}{5}$ rather than $\frac{k}{2}$, which is found in previous schemes. By their analysis, this provides a security level of $2^{-0.32k}$. Since our scheme is built on the implementation by Kreuter et al. [Kreuter et al. 2012], which uses the same cut-&-choose parameter as shelat and Shen, our garbled circuit check also provides a security parameter of $2^{-0.32k}$ which is non-polynomial and negligible for large k.

CLAIM 2. *Proof-of-work: Assuming the hash function is one-way and collision resistant, the Cloud has a negligible probability of producing a check hash that passes the seed check without actually generating the check circuit.*

As previously stated, before the circuit check begins the generator Bob sends the evaluator Alice k hashed circuit values $H_1(GC_i)$. Once the evaluation circuits are selected, the Cloud must generate λ circuits and hash them into check hashes $H_1(GC'_i)$. We assume that the Cloud does not collude with the generator (i.e. share any of the hash values sent to Alice). If the Cloud attempts to skip the generation of the check circuits, it must generate hash values $H'_i = H_i$ for $i \in Chk$. Based on security guarantees of the hash, the Cloud has a negligible probability of correctly generating these hash values.

5.2. Validity of Evaluator Inputs

To assure that the generator cannot learn anything about the evaluator's inputs by corrupting the garbled values sent during the OT, we employ the random input encoding technique by Lindell and Pinkas [Lindell and Pinkas 2007], which is built into the implementation by Kreuter et al. [Kreuter et al. 2012]. This technique allows the evaluator to encode each input bit as the XOR of a set of input bits. Thus, if the generator corrupts one of those input bits as in a selective failure attack, it reveals essentially nothing about the evaluator's true input. Additionally, we use the commitment technique employed by Kreuter et al. [Kreuter et al. 2012] to ensure that Bob cannot swap garbled input wire labels between the zero and one value. To accomplish this, the generator commits to the wire labels before the cut-&-choose. During the cut and choose, the input labels for the check circuits are opened to ensure that they correspond to only one value across all circuits. Then, during the OOT, the commitment keys for the labels that will be evaluated are sent instead of the wire labels themselves. Because our protocol implements this technique directly from the literature, we do not make any additional claims of security.

5.3. Input Consistency

The security of our input consistency check is based on two schemes, one for the evaluator's input and one for the generator's input. To assure the evaluator's inputs are consistent across circuits, we use the approach from Lindell and Pinkas [Lindell and Pinkas 2007], which is built into the implementation of Kreuter et al. [Kreuter et al. 2012]. Since the evaluator only performs one oblivious transfer for all the evaluation circuits, her received garbled inputs will all represent her input to the OT.

To assure that the generator's inputs are consistent, we employ the malleable claw-free collection approach from shelat and Shen [shelat and Shen 2011]. However, we modify the zero-knowledge proof to provide some guarantee that the Cloud server actually possesses well-formed inputs:

CLAIM 3. *Assuming the witness-indistinguishable proof used in the malleable claw-free collection input check is secure, the generator in our protocol cannot trick the evaluator into using different inputs for different evaluation circuits with greater than negligible probability.*

During the witness-indistinguishable proof, the generator sends the modified pre-image values to the mobile device, while the Cloud server sends the garbled input values of each evaluation circuit to the mobile device. The device then checks that all input values for each individual input wire were generated by the same function in the malleable claw-free pair. Based on the assumption that the generator and the Cloud will not collude, the probability of a malicious

generator providing inconsistent modified pre-image values that match the garbled inputs possessed by the Cloud server is negligible in the security parameter of the malleable claw-free pair.

5.4. Output Consistency

To ensure that the Cloud cannot learn either party's output or tamper with either party's output from the garbled circuit, we implement the technique of blinding and proving the garbled output values from the protocol by Kiraz [Kiraz and Schoenmakers 2006]. The privacy and correctness of the generator's output is guaranteed based on the security of this construction in Kiraz's two-party secure function evaluation protocol. By the same proof for the generator's output remaining secure, we argue that the evaluator's output is also secure and correct. By using the same construction for both parties' outputs, we guarantee output privacy and consistency, even in the presence of a malicious Cloud. Note that to maintain security, this construction only provides Bob and Alice with the output of computation, not the index of the majority evaluation circuit.

5.5. Outsourced Oblivious Transfer

Our outsourced oblivious transfer is an extension of a technique developed by Naor et al. [Naor et al. 1999] that allows the chooser to select entries that are forwarded to a third party rather than returned to the chooser. By combining their concept of a proxy oblivious transfer with the semi-honest OT extension by Ishai et al. [Ishai et al. 2003], our outsourced oblivious transfer provides a secure OT in the malicious model. We achieve this result for four reasons:

(1) First, since Alice never sees the outputs of the OT protocol, she cannot learn anything about the garbled values held by the generator. This saves us from having to implement Ishai's extension to prevent the chooser from behaving maliciously.
(2) Since the Cloud sees only random garbled values and Alice's input blinded by a one-time pad, the Cloud learns nothing about Alice's true inputs.
(3) Since Bob's view of the protocol is almost identical to his view in Ishai's standard extension, the same security guarantees hold (i.e., security against a malicious sender).
(4) Finally, if Alice does behave maliciously and uses inconsistent inputs to the primitive OT phase, there is a negligible probability that those values will hash to the correct one-time pad keys for recovering *either* commitment key, which will prevent the Cloud from de-committing the garbled input values.

It is important to note that this particular application of the OOT allows for this efficiency gain since the evaluation of the garbled circuit will fail if Alice behaves maliciously. By applying the maliciously secure extension by Ishai et al. [Ishai et al. 2003], this primitive could be applied generally as an oblivious transfer primitive that is secure in the malicious model. Further discussion and analysis of this general application is outside the scope of this work.

6. PROOF OF SECURITY

We formally prove the security of our protocol with the following theorem, which gives security guarantees identical to the Salus protocol by Kamara et al. [Kamara et al. 2012].

THEOREM 1. *The outsourced two-party SFE protocol securely computes a function $f(a, b)$ in the following two corruption scenarios: (1)The Cloud is malicious and non-cooperative with respect to the rest of the parties, while all other parties are semi-honest, (2)All but one party is malicious, while the Cloud is semi-honest.*

PROOF. To demonstrate that:

$$\{REAL^{(i)}(k, x; r)\}_{k\in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k\in N}$$

for all $i \in [A, B, C]$, we consider separately the cases when Alice, Bob, and Cloud deviate from the protocol.

6.1. Malicious evaluator Alice A^*

In this scenario, both Bob and Cloud participate honestly in the protocol. Note that during the protocol execution, Alice only exchanges messages with the other participants at five points: the coin-flip during the cut-&-choose, the primitive oblivious transfer, sending decryption information at the end of the OOT, checking Bob's input consistency, and receiving the proof of validity and output from the garbled circuit. Thus, our simulator need only ensure that these sections of the protocol are indistinguishable to the adversary A^*. Consider the following hybrid experiments and lemmas.

Simulating the coin-flip (Phase 1):

$\boldsymbol{Hybrid1^{(A)}(k, x; r)}$: This experiment is the same as $REAL^{(A)}(k, x; r)$ except that instead of running a fair coin toss protocol with A^*, the experiment chooses a random string ρ, and a coin-flipping simulator $S_{CF}(\rho, 1^k)$ produces the protocol messages that output ρ.

LEMMA 6.1. $REAL^{(A)}(k, x; r) \overset{c}{\approx} Hybrid1^{(A)}(k, x; r)$

PROOF. Based on the security of the fair coin toss protocol, we know that there exists a simulator $S_{CF}(\cdot,\cdot)$ such that an interaction with $S_{CF}(\cdot,\cdot)$ is indistinguishable from a real protocol interaction. Since everything else in $Hybrid1^{(A)}(k,x;r)$ is exactly the same as in $REAL^{(A)}(k,x;r)$, this proves the lemma. □

Simulating the primitive OT (Phase 2):

$\boldsymbol{Hybrid2^{(A)}(k,x;r)}$: This experiment is the same as $Hybrid1^{(A)}(k,x;r)$ except that during the Outsourced Oblivious Transfer, the experiment invokes a simulator S_{OT} to simulate the primitive oblivious transfer operation with A^*. The simulator sends A^* a random string s and receives the columns of the matrix Q^*.

LEMMA 6.2. $Hybrid1^{(A)}(k,x;r) \overset{c}{\approx} Hybrid2^{(A)}(k,x;r)$

PROOF. Based on the malicious security of the OT primitive, we know that there exists a simulator S_{OT} such that an interaction with this simulator is indistinguishable from a real execution of the oblivious transfer protocol. Since everything else in $Hybrid2^{(A)}(k,x;r)$ is identical to $Hybrid1^{(A)}(k,x;r)$, this proves the lemma. □

Checking the output of OOT (Phase 2):

$\boldsymbol{Hybrid3^{(A)}(k,x;r)}$: This experiment is the same as $Hybrid2^{(A)}(k,x;r)$ except that the experiment aborts if the matrix Q^* is not formed correctly (that is, if A^* used inconsistent input values ea^* for any column in generating Q^*).

LEMMA 6.3. $Hybrid2^{(A)}(k,x;r) \overset{c}{\approx} Hybrid3^{(A)}(k,x;r)$

PROOF. Consider that in $Hybrid2^{(A)}(k,x;r)$, if for some value of i, A^* sends the column value $T^i \oplus ea'$ for some $ea' \neq ea^*$ such that the i^{th} bit is b in ea^* and $b \oplus 1$ in ea'. Then for every row in Q^*, the i^{th} bit will be encrypted in the $b \oplus 1$ entry. However, when A^* sends the value $ea^* \oplus p^*$ to the Cloud for decryption, when the Cloud decrypts the i^{th} choice, it will decrypt the $b \oplus 1$ entry instead of the b entry, which will yield an invalid decryption with probability $1-\epsilon$ for a negligible value of ϵ. Since, with high probability, this decryption is not a valid commitment key, the garbled input values will not decommit properly and the Cloud will abort. In $Hybrid3^{(A)}(k,x;r)$, since the experiment observes the messages Q^*, p^*, and $ea^* \oplus p^*$, it can recover ea^* and check Q^* for consistency, aborting if an inconsistency is found. □

Simulating consistency check and substituting inputs (Phase 3):

$\boldsymbol{Hybrid4^{(A)}(k,x;r)}$: This experiment is the same as $Hybrid3^{(A)}(k,x;r)$ except that the experiment provides a string of $2 \cdot n$ zeros, denoted $\{0\}^{2 \cdot n}$, during the consistency check to replace Bob's input b.

LEMMA 6.4. $Hybrid3^{(A)}(k,x;r) \overset{c}{\approx} Hybrid4^{(A)}(k,x;r)$

PROOF. Here we cite Lemma 5 from the proof of shelat and Shen's scheme [shelat and Shen 2011]. Since the messages sent in our scheme are identical to theirs in content, we simply change the entity sending the message in the experiment and the lemma still holds. □

Simulating the output proof (Phase 5):

$\boldsymbol{Hybrid5^{(A)}(k,x;r)}$: This experiment is the same as $Hybrid4^{(A)}(k,x;r)$ except that instead of returning the output of the circuit, the experiment provides A^* with the result sent from the trusted external oracle.

LEMMA 6.5. $Hybrid4^{(A)}(k,x;r) \overset{c}{\approx} Hybrid5^{(A)}(k,x;r)$

PROOF. Based on the security of the garbled circuit construction being used, the trusted third party output and the circuit output will be indistinguishable when provided with A^*s input ea^*, which the experiment can recover because of the change made in $Hybrid3^{(A)}(k,x;r)$. In addition, since the experiment can observe the random seeds used to construct the proofs of output consistency used when generating the evaluation circuits, the experiment can reproduce valid proofs of consistency for the output value $f_A(a^*,b)$ provided by the oracle. Based on the security proofs of these consistency checks by Kiraz et al. [Kiraz and Schoenmakers 2006], indistinguishability holds. □

LEMMA 6.6. $Hybrid5^{(A)}(k,x;r)$ *runs in polynomial time.*

PROOF. Since A^* is strictly a polynomial-time adversary and all of the operations in the $REAL$ protocol are polynomial time, most of the operations performed can be summarized into a runtime $p(k)$. The two simulators S_{OT} and $S_{CF}(\cdot,\cdot)$ are assumed to be polynomial in runtime since they are, computationally secure simulators for their respective roles. Since they are both only executed once, the total running time can be expressed as $p(k) + r_{OT} + r_{CF}$,

where r_{OT} is the runtime of the polynomial simulator S_{OT} and r_{CF} is the runtime of the polynomial simulator $S_{CF}(\cdot,\cdot)$. Since all of these individual components are polynomial, the total runtime is also polynomial. □

$Hybrid5^{(A)}(k,x;r)$ is exactly the experiment run by the simulator S^A in the ideal world. Should A^* ever abort the protocol, the simulator S^A will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid5^{(A)}(k,x;r)$, controlling Bob and Cloud, and outputs whatever A^* outputs. By Lemma 6.1-6.6, this simulator proves Theorem 1 when Alice is malicious.

6.2. Malicious generator Bob B^*

In this scenario, both Alice and Cloud participate honestly in the protocol. Note that in the protocol, the generator exchanges messages with both parties at six critical points: circuit cut-&-choose, the primitive OT, the OOT result delivery, the input consistency check, the circuit pipelined evaluation, and the output proof of integrity and delivery. Consider the following hybrid experiments and lemmas.

Simulating the cut-&-choose (Phase 1):

$\boldsymbol{Hybrid1^{(B)}(k,x;r)}$: This experiment is the same as $REAL^{(B)}(k,x;r)$ except that if B^* successfully passes the first cut-&-choose test, the experiment repeatedly rewinds B^*, repeating the coin flip protocol and the verification of the circuit hashes and commitments until B^* passes for a second time. Let Chk_i be the set of check circuit indices for the i^{th} successful cut-&-choose. If $Chk_1 = Chk_2$, then the experiment aborts.

LEMMA 6.7. $REAL^{(B)}(k,x;r) \overset{c}{\approx} Hybrid1^{(B)}(k,x;r)$

PROOF. Here we cite Lemma 8 from shelat and Shen's protocol proof [shelat and Shen 2011]. The idea in their proof is that if B^* never passes the cut-&-choose, then both the real and hybrid experiments will abort. However, if B^* passes once, they demonstrate that the probability of B^* passing again with the exact same set of check circuits is negligible within a polynomial number of rewinds. Since their cut-&-choose is identical to ours, indistinguishability holds in this setting. □

Simulating the primitive OT (Phase 2):

$\boldsymbol{Hybrid2^{(B)}(k,x;r)}$: This experiment is the same as $Hybrid1^{(B)}(k,x;r)$ except that rather than run the primitive oblivious transfer with Alice, the experiment generates a random input string ea' and a random matrix T, then runs a simulator S_{OT} with B^*, which delivers to B^* exactly one element from the pair $(T^i, T^i \oplus ea')$ depending on B^*'s i^{th} selection bit.

LEMMA 6.8. $Hybrid1^{(B)}(k,x;r) \overset{c}{\approx} Hybrid2^{(B)}(k,x;r)$

PROOF. Based on the security of the primitive OT scheme, we know that the simulator S_{OT} exists, that it can recover B^*'s selection bits s^* from the interaction, and that an interaction with it is indistinguishable from a real execution of the OT. Since B^* cannot learn any distinguishing information from Alice's input, again based on the security of the OT primitive, then indistinguishability holds between the hybrid experiments. □

Checking the output of OOT (Phase 2):

$\boldsymbol{Hybrid3^{(B)}(k,x;r)}$: This experiment is the same as $Hybrid2^{(B)}(k,x;r)$ except that the experiment checks the validity of B^*'s output from the OOT. Since the experiment possesses T, ea', and s^* (which was recovered by the oblivious transfer simulator S_{OT} in the previous hybrid), the experiment can check whether or not the encrypted set of outputs Y^* is well-formed. If it is not, the experiment immediately aborts.

LEMMA 6.9. $Hybrid2^{(B)}(k,x;r) \overset{c}{\approx} Hybrid3^{(B)}(k,x;r)$

PROOF. Recall that in $Hybrid2^{(B)}(k,x;r)$, if B^* does not format the output of the OOT correctly, Cloud will, with probability $1-\epsilon$ fail to recover a valid commitment key, where ϵ is negligible in the security parameter. Should this be the case, the committed garbled circuit labels will fail to decrypt properly, and the Cloud will abort the protocol. In $Hybrid3^{(B)}(k,x;r)$, since the experiment has observed the values Q, s^*, ea', and Y^*, it can trivially observe if Y^* is correctly formed, and aborts if it is not. Additionally, for B^* to swap any of A's input labels in the commitments, B^* must find a claw in the claw-free collection used to generate those input labels. Based on the security of shelat and Shen's claw-free collections technique, used to check the consistency of A's input across evaluation circuits in this phase, this will only happen with a negligible probability. □

Checking input consistency and recovering inputs (Phase 3):

$\boldsymbol{Hybrid4^{(B)}(k,x;r)}$: This experiment is the same as $Hybrid3^{(B)}(k,x;r)$ except that the experiment recovers B^*'s input b^* during the input consistency check using the random seed recovered in $Hybrid1^{(B)}(k,x;r)$. Since the experiment is running with a set of evaluation circuits Evl_2 produced during the second successful cut-&-choose, it possesses the random coins used to generate at least one of these circuits since $Evl_1 \neq Evl_2$. If the consistency check does not pass or if B^*'s input cannot be recovered, the experiment immediately aborts.

LEMMA 6.10. $Hybrid3^{(B)}(k,x;r) \stackrel{c}{\approx} Hybrid4^{(B)}(k,x;r)$

PROOF. Based on the security of shelat and Shen's claw-free collections technique for checking the consistency of B^*'s inputs across evaluation circuits, then B^* can find a claw and change its input for one evaluation circuit with negligible probability. In $Hybrid4^{(B)}(k,x;r)$, since the experiment possesses at least one set of random coins rc_i that was used to generate a circuit in the set Evl_2, it can recover the input wire labels for that circuit and recover B^*'s input. If the input is invalid (i.e., does not correspond to an input label), the circuit would fail to evaluate and would generate an abort in both hybrids. Thus, indistinguishability holds. □

Simulating the output proof (Phase 5):

$\boldsymbol{Hybrid5^{(B)}(k,x;r)}$: This experiment is the same as $Hybrid4^{(B)}(k,x;r)$ except that during the output phase the experiment prepares the result received from the trusted third party as the output instead of the output from the circuit. If no majority values fb' is found from the circuit, the experiment aborts. Otherwise, it uses the random coins rc_i recovered from $Hybrid1^{(B)}(k,x;r)$ to prove the validity of the output for some circuit.

LEMMA 6.11. $Hybrid4^{(B)}(k,x;r) \stackrel{c}{\approx} Hybrid5^{(B)}(k,x;r)$

PROOF. Based on the security of the garbled circuit construction being used, the trusted third party output and the circuit output will be indistinguishable when provided with B^*'s input b^*, which the experiment can recover using the random coins rc_i obtained in $Hybrid1^{(B)}(k,x;r)$. In addition, these random coins allow the experiment to construct the proofs of output consistency used when generating one of the evaluation circuits, thus the experiment can reproduce valid proofs of consistency for the output value $f_B(a,b^*)$ provided by the trusted third party. If no majority output value exists, in both hybrids the abort message would be sent. Finally, based on the security proofs of the witness indistinguishable consistency proofs developed by Kiraz et al. [Kiraz and Schoenmakers 2006], indistinguishability holds. □

LEMMA 6.12. $Hybrid5^{(B)}(k,x;r)$ *runs in polynomial time.*

PROOF. Since all of the steps in the protocol run in expected polynomial time, the only step that must be verified is the rewinding phase. Based on lemma 14 from shelat and Shen's proof [shelat and Shen 2011], the total time for the rewinds is also polynomial in k. Thus, the composition of all steps runs in polynomial time. □

$Hybrid5^{(B)}(k,x;r)$ is exactly the experiment run by the simulator S^B in the ideal world. Should B^* ever abort the protocol, the simulator S^B will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid5^{(B)}(k,x;r)$, controlling Alice and Cloud, and outputs whatever B^* outputs. By Lemma 6.7-6.12, this simulator proves Theorem 1 when Bob is malicious.

6.3. Malicious Cloud C^*

In this scenario, both Alice and Bob participate honestly in the protocol. Note that in the protocol, the Cloud participates in checking the circuits during the cut-&-choose, decrypting Alice's inputs in the OOT, forwarding Bob's inputs for consistency checking, evaluating the circuit, and proving and delivering the final output of computation. Consider the following hybrid experiments and lemmas.

Replacing inputs for the OOT (Phase 2):

$\boldsymbol{Hybrid1^{(C)}(k,x;r)}$: This experiment is the same as $REAL^{(C)}(k,x;r)$ except that during the OOT, the experiment replaces Alice's input ea with a string of zeros $ea' = \{0\}^{2\cdot\ell\cdot n}$. This value is then used to select garbled input values from Bob in the OOT, which are then forwarded to C^* according to the protocol

LEMMA 6.13. $REAL^{(C)}(k,x;r) \stackrel{c}{\approx} Hybrid1^{(C)}(k,x;r)$

PROOF. In a real execution, C^* will observe the random matrix T, the encrypted commitment keys Y, and Alice's input XOR'd with the permutation string $ea \oplus p$. Based on the statistical indistinguishability of a value XOR'd with a random value, $ea \oplus p \stackrel{s}{\approx} p$ for any input value ea. Since T is randomly generated in both $REAL^{(C)}(k,x;r)$ and $Hybrid1^{(C)}(k,x;r)$, they are trivially indistinguishable. Considering the output pairs Y, half of the commitment keys

(those not selected by Alice) will consist of the keys in $x_{b,j} \oplus H(j,s)$, which is computationally indistinguishable from random, and the keys in $x_{b,j}$ can only be recovered if C^* can find a collision with the hash value $H(j,s)$ without having Bob's random value s. The remaining keys, which can be recovered by C^*, are permuted randomly, such that their ordering is statistically indistinguishable from a random ordering. Since the commitments are also permuted randomly, the same indistinguishability holds for the ordering of the garbled input wire values. Thus, C^* cannot distinguish an execution of OOT with Alice's input ea and the simulator's input replacement ea'. Since the rest of the protocol follows $REAL^{(C)}(k,x;r)$ exactly, this proves the lemma. □

Replacing inputs for the consistency check (Phase 3):

$\boldsymbol{Hybrid2^{(C)}(k,x;r)}$: This experiment is the same as $Hybrid1^{(C)}(k,x;r)$ except that the experiment replaces Bob's input b with all zeros $\{0\}^{2\cdot n}$. This value is then prepared and checked according to the protocol for consistency across evaluation circuits.

LEMMA 6.14. $Hybrid1^{(C)}(k,x;r) \overset{c}{\approx} Hybrid2^{(C)}(k,x;r)$

PROOF. In this hybrid, C^* observes a set of garbled input wire values from Bob. Based on the security of Yao garbled circuits, observing one set of garbled input wire values is indistinguishable from observing any other set of input wire values, such that C^* cannot distinguish between the garbled input for b and the garbled input for $\{0\}^{2\cdot n}$. Since the rest of the hybrid is the same as $Hybrid1^{(C)}(k,x;r)$, this proves the lemma. □

Checking the output of the circuit (Phase 5):

$\boldsymbol{Hybrid3^{(C)}(k,x;r)}$: This experiment is the same as $Hybrid2^{(C)}(k,x;r)$ except that after the circuit is evaluated, the experiment checks that the results output by C^* matches the expected results $f_A(a',b')$ and $f_B(a',b')$. If C^* fails to produce a valid proof that the output came from the circuit or if the result does not match, the experiment immediately aborts.

LEMMA 6.15. $Hybrid2^{(C)}(k,x;r) \overset{c}{\approx} Hybrid3^{(C)}(k,x;r)$

PROOF. In $Hybrid2^{(C)}(k,x;r)$, C^* can only modify the output without detection with probability $1-\epsilon$ for some negligible probability ϵ, based on the security guarantees of Kiraz's proof scheme [Kiraz and Schoenmakers 2006]. In $Hybrid3^{(C)}(k,x;r)$ the experiment catches C^* when it tries to change the output of the circuit with probability 1, so the distributions are computationally indistinguishable. □

LEMMA 6.16. $Hybrid3^{(C)}(k,x;r)$ *runs in polynomial time.*

PROOF. Since the protocol itself requires only polynomially many steps, the time to run $REAL^{(C)}(k,x;r)$ can be expressed as $p(k)$. The experiment also evaluates the polynomial-time function $f(\cdot,\cdot)$ over random inputs to check the output of C^*. We call this execution time $q(c)$, where c is the size of the circuit being evaluated. So, the total execution time is $p(k)+q(c)$, which is polynomial as a sum. □

$Hybrid3^{(C)}(k,x;r)$ is exactly the experiment run by the simulator S^C in the ideal world. Should C^* ever abort the protocol, the simulator S^C will forward the abort to the trusted third party. Otherwise, it will follow $Hybrid3^{(C)}(k,x;r)$, controlling Alice and Bob. If the cut-&-choose, input consistency check, or output proof of correctness fail, then S^C aborts to both C^* and the trusted third party. Otherwise, S^C outputs whatever C^* outputs. By Lemma 6.13-6.16, this simulator proves Theorem 1 when Cloud is malicious.

Given the simulators S_A, S_B, and S_C, this proves the security of our protocol as stated in Theorem 1.

7. PERFORMANCE ANALYSIS

We now characterize how garbled circuits perform in the constrained-mobile environment with and without outsourcing. Two of the most important constraints for mobile devices are computation and bandwidth, and we show that order of magnitude improvements for both factors are possible with outsourced evaluation[1]. We begin by describing our implementation framework and testbed before discussing results in detail. These results extend the experimental work performed by Carter et al. [**?**].

7.1. Framework and Testbed

Our framework is based on the system designed by Kreuter et al. [Kreuter et al. 2012], hereafter referred to as *KSS* for brevity. We implemented the outsourced protocol and performed modifications to allow for the use of the mobile device

[1]We contacted the authors of the Salus protocol [Kamara et al. 2012] in an attempt to acquire their framework to compare the actual performance of their scheme with ours, but they were unable to release their code. Moreover, the authors said that the results reported in their work were not accurate, so no sound comparison to their work was possible.

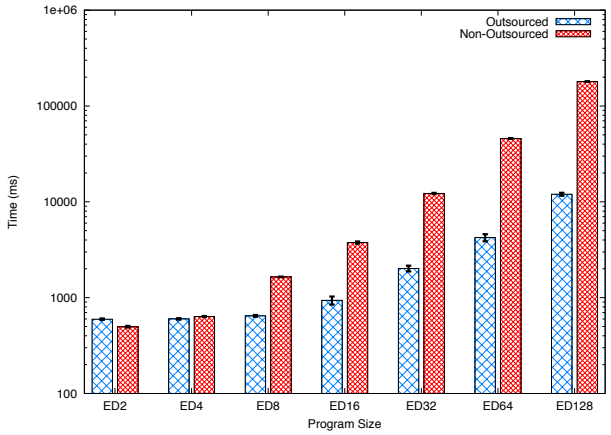


Fig. 3: Execution time for the Edit Distance program of varying input sizes, with 2 circuits evaluated. On 12 core servers.

in the computation. Notably, *KSS* uses MPI for communication between the multiple nodes of the multi-core machines relied on for circuit evaluation. Our solution replaces MPI calls on the mobile device with sockets that communicate directly with the Generator and Proxy. To provide a consistent comparison, we revised the *KSS* codebase to allow for direct evaluation between the mobile device (the Evaluator) and the cloud-based Generator. The modifications required included removing the MPI library from the phone client and functions, which did not exist on the phone. The largest difficulty was changing the Intel specific instructions to generic instruction, which would work on the ARM processor of the mobile device.

We also informed the original authors of *KSS* of several problems we noticed. They in turn fixed the problems necessary for our trials. We found the following problems: generator's input consistency check was missing from one of the possible run environments, arrays did not work correctly in some cases, nested if statements did not work correctly, and for loops inside of if statements did not work correctly. We thank those authors for their assistance.

Our deployment platform consists of two Dell R610 servers, each containing dual 6-core Xeon processors with 32 GB of RAM and 300 GB 10K RPM hard drives, running the Linux 3.4 kernel and connected as a VLAN on an internal 1 Gbps switch. These machines perform the roles of the Generator and Proxy, respectively, as described in Section 4.1. The mobile device acts as the Evaluator. We use a Samsung Galaxy Nexus phone with a 1.2 GHz dual-core ARM Cortex-A9 processor and 1 GB of RAM, running the Android 4.0 "Ice Cream Sandwich" operating system. We connect an Apple Airport Express wireless access point to the switch attaching the servers, The Galaxy Nexus communicates to the Airport Express over an 802.11n 54Mbps WiFi connection in an isolated environment to minimize co-channel interference. All tests are run 10 times with error bars on figures representing 95% confidence intervals.

We also ran tests using a 64 core server with 1 TB of memory where the phone was connected over a standard local network, which included additional traffic other than just our tests. This platform performed the roles of both the Generator and the Proxy.

7.2. Execution Time

Our tests evaluated the following problems:

Millionaires: This problem models the comparison of two parties comparing their net worth to determine who has more money without disclosing the actual values. We perform the test on input values ranging in size from 4 to 8192 bits.

Edit (Levenshtein) Distance: This is a string comparison algorithm that compares the number of modifications required to covert one string into another. We performed the comparison based on the circuit generated by Jha et al. [Jha et al. 2008] for strings sized between 4 and 128 bytes.

Set Intersection: This problem matches elements between the private sets of two parties without learning anything beyond the intersecting elements. We base our implementation on the SCS-WN protocol proposed by Huang et al. [Huang et al. 2012], and evaluate for sets of size 2 to 128.

AES: We compute AES with a 128-bit key length, based on a circuit evaluated by Kreuter et al. [Kreuter et al. 2012].

Figure 3 shows the result of the edit distance computation for input sizes of 2 to 128 with two circuits evaluated. This comparison represents worst-case operation due to the cost of setup for a small number of small circuits - with input size 2, the circuit is only 122 gates in size. For larger input sizes, however, outsourced computation becomes significantly faster. Note that the graph is logarithmic such that by the time strings of size 32 are evaluated, the

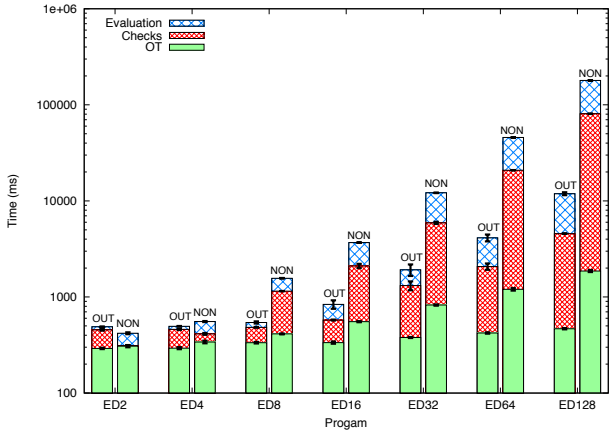


Fig. 4: Execution time for significant stages of garbled circuit computation for outsourced and non-outsourced evaluation. The Edit Distance program is evaluated with variable input sizes for the two-circuit case. On 12 core servers.

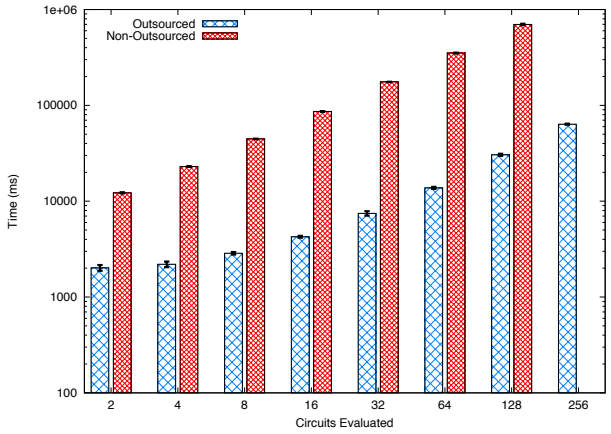


Fig. 5: Execution time for the Edit Distance problem of size 32, with between 2 and 256 circuits evaluated. In the non-outsourced evaluation scheme, the mobile phone runs out of memory evaluating 256 circuits. On 12 core servers.

outsourced execution is over 6 times faster than non-outsourced execution, while for strings of size 128 (comprising over 3.4 million gates), outsourced computation is over 16 times faster.

The reason for this becomes apparent when we examine Figure 4. There are three primary operations that occur during the SFE transaction: the oblivious transfer (OT) of participant inputs, the circuit commit (including the circuit consistency check), and the circuit evaluation. As shown in the figure, the OT phase takes 292 ms for input size 2, but takes 467 ms for input size 128. By contrast, in the non-outsourced execution, the OT phase takes 307 ms for input size 2, but increases to 1860 ms for input size 128. The overwhelming factor, however, is the circuit evaluation phase. It increases from 34 ms (input size 2) when the evaluation is complete by the time the checks finish on the phone to 7320 ms (input size 128) for the outsourced evaluation, a 215 factor increase. For non-outsourced execution however, this phase increases from 108 ms (input size 2) to 98800 ms (input size 128), a factor of 914 increase.

7.3. Evaluating Multiple Circuits

The security parameter for the garbled circuit check is $2^{-0.32k}$ [Kreuter et al. 2012], where k is the number of generated circuits. To ensure a sufficiently low probability (2^{-80}) of evaluating a corrupt circuit, 256 circuits must be evaluated. However, there are increasing execution costs as increasing numbers of circuits are generated. Figure 5 shows the execution time of the Edit Distance problem of size 32 with between 2 and 256 circuits being evaluated. In the outsourced scheme, costs rise as the number of circuits evaluated increases. Linear regression analysis shows we can model execution time T as a function of the number of evaluated circuits k with the equation $T = 243.2k + 334.6$ ms, with a coefficient of determination R^2 of 0.9971. However, note that in the non-outsourced scheme, execution time increases over 10 times as quickly compared to outsourced evaluation. Regression analysis shows execution time $T = 5435.7k + 961$ ms, with $R^2 = 0.9998$. Because in this latter case, the mobile device needs to perform all

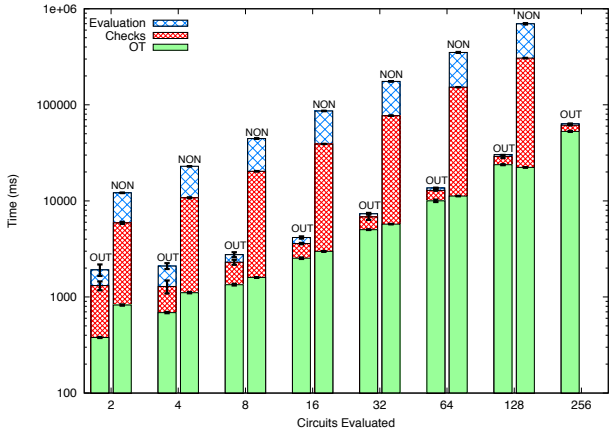


Fig. 6: Microbenchmarks of execution time for Edit Distance with input size 32, evaluating from 2 to 256 circuits. Note that the y-axis is log-scale; consequently, the vast majority of execution time is in the check and evaluation phases for non-outsourced evaluation. On 12 core servers.

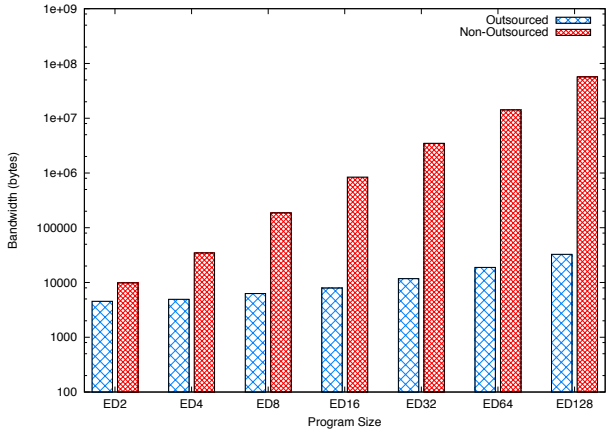


Fig. 7: Bandwidth measurements from the phone to remote parties for the Edit Distance problem with varying input sizes, executing two circuits. On 12 core servers.

computation locally as well as transmit all circuit data to the remote parties, these costs increase rapidly. Figure 6 provides more detail about each phase of execution. Note that the OT costs are similar between outsourced and non-outsourced execution for this circuit size, but that the costs of consistency checks and evaluation vastly increase execution time for non-outsourced execution.

Note as well that in the non-outsourced scheme, there are no reported values for 256 circuits, as the Galaxy Nexus phone ran out of memory before the execution completed. We observe that a single process on the phone is capable of allocating 512 MB of RAM before the phone would report an out of memory error, providing insight into how much intermediate state is required for non-outsourced evaluation. Thus, to handle circuits of any meaningful size with enough check circuits for a strong security parameter, the *only way* to be able to perform these operations is through outsourcing.

Tables I and II present the execution time of the circuits we evaluated. It spans circuits from small to large input size, and from 8 circuits evaluated to the 256 circuits required for a 2^{-80} security parameter. Note that in many cases it is impossible to evaluate the non-outsourced computation because of the mobile device's inability to store sufficient amounts of state. Note as well that particularly with complex circuits such as set intersection, even when the non-outsourced evaluation is capable of returning an answer, it can require orders of magnitude more time than with outsourced evaluation. For example, evaluating the set intersection problem with 128 inputs over 32 circuits requires just over 55 seconds for outsourced evaluation but *over an hour and a half* with the non-outsourced *KSS* execution scheme. Outsourced evaluation represents a time savings of 98.92%. Tables III and IV reflect the 64 core results for the same set of tests. We also ran the non-outsourced *KSS* execution scheme on this server as a comparison point.

	2 Circuits		4 Circuits		8 Circuits	
Program	Outsourced	KSS	Outsourced	KSS	Outsourced	KSS
Millionaires 4	553.0 ± 2%	379.0 ± 3%	732.0 ± 1%	443.0 ± 2%	1090.0 ± 1%	536.0 ± 2%
Millionaires 32	667.0 ± 3%	953.0 ± 2%	985.0 ± 2%	1260.0 ± 3%	1620.0 ± 2%	1811.0 ± 0.8%
Millionaires 128	821.0 ± 2%	2180.0 ± 1%	1230.0 ± 1%	3430.0 ± 1%	2150.0 ± 1%	6130.0 ± 0.6%
Millionaires 1024	2200.0 ± 10%	13550.0 ± 0.8%	2820.0 ± 8%	23990.0 ± 0.7%	4670.0 ± 6%	46290.0 ± 0.4%
Millionaires 8192	6050.0 ± 2%	99160.0 ± 0.4%	10400.0 ± 1%	185500.0 ± 0.5%	17280.0 ± 0.9%	368800.0 ± 0.4%
Edit Distance 2	596.0 ± 2%	497.0 ± 2%	803.0 ± 2%	625.0 ± 2%	1268.0 ± 0.9%	794.0 ± 1%
Edit Distance 4	600.0 ± 2%	636.0 ± 1%	810.0 ± 2%	914.0 ± 2%	1260.0 ± 2%	1770.0 ± 9%
Edit Distance 8	646.0 ± 2%	1646.0 ± 0.7%	912.0 ± 2%	2280.0 ± 5%	1480.0 ± 2%	3590.0 ± 2%
Edit Distance 16	940.0 ± 10%	3760.0 ± 3%	1180.0 ± 2%	6370.0 ± 2%	1730.0 ± 2%	11900.0 ± 1%
Edit Distance 32	2020.0 ± 7%	12200.0 ± 1%	2200.0 ± 7%	23010.0 ± 0.6%	2860.0 ± 3%	44610.0 ± 0.7%
Edit Distance 64	4230.0 ± 8%	45810.0 ± 0.8%	4650.0 ± 6%	89680.0 ± 0.5%	5070.0 ± 4%	176300.0 ± 0.7%
Edit Distance 128	12000.0 ± 4%	179700.0 ± 0.6%	12000.0 ± 3%	354500.0 ± 0.2%	12800.0 ± 2%	702400.0 ± 0.5%
Set Intersection 2	655.0 ± 1%	997.0 ± 2%	983.0 ± 3%	1330.0 ± 4%	1598.0 ± 0.8%	1856.0 ± 0.9%
Set Intersection 4	773.0 ± 2%	1700.0 ± 3%	1090.0 ± 3%	2740.0 ± 5%	1870.0 ± 2%	3970.0 ± 4%
Set Intersection 8	952.0 ± 3%	3698.8 ± 3%	1300.0 ± 4%	5644.9 ± 2%	2258.5 ± 2%	9286.6 ± 0.9%
Set Intersection 16	2070.0 ± 5%	8830.0 ± 2%	2120.0 ± 4%	15360.0 ± 0.8%	3400.0 ± 10%	28090.0 ± 0.7%
Set Intersection 32	3400.0 ± 10%	27300.0 ± 1%	3400.0 ± 5%	50200.0 ± 0.6%	5200.0 ± 10%	96560.0 ± 0.6%
Set Intersection 64	7500.0 ± 2%	95930.0 ± 0.6%	7490.0 ± 4%	183500.0 ± 0.8%	8750.0 ± 5%	356100.0 ± 0.9%
Set Intersection 128	21800.0 ± 1%	360800.0 ± 0.5%	22300.0 ± 2%	703000.0 ± 0.5%	24300.0 ± 2%	1398000.0 ± 0.4%
AES-128	1110.0 ± 3%	5450.0 ± 3%	1440.1 ± 3%	8745.0 ± 0.9%	2450.0 ± 2%	15040.0 ± 0.7%

Table I: Execution time (in ms) of outsourced vs non-outsourced (KSS) evaluation for a subset of circuits. Results with a dash indicate evaluation that the phone was incapable of performing. On 12 core servers.

	32 Circuits		128 Circuits		256 Circuits	
Program	Outsourced	KSS	Outsourced	KSS	Outsourced	KSS
Millionaires 4	3320.0 ± 1%	1300.0 ± 5%	16200.0 ± 2%	3860.0 ± 0.7%	36800.0 ± 1%	7378.0 ± 0.5%
Millionaires 32	5760.0 ± 3%	6181.0 ± 0.6%	25900.0 ± 2%	23530.0 ± 0.7%	56400.0 ± 1%	46620.0 ± 0.6%
Millionaires 128	8210.0 ± 3%	23080.0 ± 0.6%	38100.0 ± 7%	91020.0 ± 0.8%	75700.0 ± 1%	180800.0 ± 0.5%
Millionaires 1024	17800.0 ± 1%	180500.0 ± 0.3%	75290.0 ± 1%	744500.0 ± 0.7%	151000.0 ± 1%	1507000.0 ± 0.5%
Millionaires 8192	76980.0 ± 0.5%	1519000.0 ± 0.4%	351300.0 ± 0.7%	-	880000.0 ± 20%	-
Edit Distance 2	4060.0 ± 1%	2125.0 ± 0.7%	19200.0 ± 2%	7476.0 ± 0.5%	42840.0 ± 0.4%	14600.0 ± 0.8%
Edit Distance 4	4070.0 ± 1%	3310.0 ± 4%	19600.0 ± 3%	11450.0 ± 0.4%	43080.0 ± 0.3%	22880.0 ± 0.6%
Edit Distance 8	4969.0 ± 0.8%	11000.0 ± 2%	22700.0 ± 1%	40910.0 ± 0.7%	49800.0 ± 1%	82920.0 ± 0.9%
Edit Distance 16	5580.0 ± 3%	43550.0 ± 0.4%	24100.0 ± 3%	176000.0 ± 0.8%	51300.0 ± 1%	354000.0 ± 1%
Edit Distance 32	7470.0 ± 5%	175600.0 ± 0.5%	30500.0 ± 3%	699000.0 ± 2%	63600.0 ± 1%	-
Edit Distance 64	12500.0 ± 3%	701000.0 ± 1%	48000.0 ± 2%	-	97340.0 ± 0.8%	-
Edit Distance 128	30300.0 ± 2%	2805000.0 ± 0.8%	106200.0 ± 0.6%	-	213400.0 ± 0.3%	-
Set Intersection 2	5720.0 ± 0.7%	6335.0 ± 0.4%	26100.0 ± 2%	24420.0 ± 0.6%	56350.0 ± 0.8%	48330.0 ± 0.6%
Set Intersection 4	6797.0 ± 0.7%	13980.0 ± 0.6%	30500.0 ± 2%	55040.0 ± 0.6%	64410.0 ± 0.8%	110800.0 ± 0.5%
Set Intersection 8	8170.0 ± 1%	33840.0 ± 0.5%	35600.0 ± 2%	154100.0 ± 0.5%	76400.0 ± 1%	332500.0 ± 0.5%
Set Intersection 16	10600.0 ± 6%	107800.0 ± 0.6%	44500.0 ± 1%	501000.0 ± 0.6%	91690.0 ± 0.8%	1033000.0 ± 0.6%
Set Intersection 32	13800.0 ± 1%	400800.0 ± 0.6%	59400.0 ± 1%	-	125300.0 ± 0.9%	-
Set Intersection 64	22600.0 ± 1%	1493300.0 ± 0.7%	93000.0 ± 1%	-	187300.0 ± 1%	-
Set Intersection 128	55400.0 ± 3%	5712000.0 ± 0.4%	1998000.0 ± 0.5%	-	395200.0 ± 0.8%	-
AES-128	9090.0 ± 5%	58920.0 ± 0.5%	39000.0 ± 2%	276200.0 ± 0.6%	81900.0 ± 1%	577900.0 ± 0.5%

Table II: Execution time (in ms) of outsourced vs non-outsourced (KSS) evaluation for a subset of circuits. Results with a dash indicate evaluation that the phone was incapable of performing. On 12 core servers.

Multicore Circuit Evaluation. We briefly note the effects of multicore servers for circuit evaluation. The servers in our evaluation each contain dual 6-core CPUs, providing 12 total cores of computation. The computation process is largely CPU-bound: while circuits on the servers are being evaluated, each core was reporting approximately 100% utilization. This is evidenced by regression analysis when evaluating between 2 and 12 circuit copies; we find that execution time $T = 162.6k + 1614.6$ ms, where k is the number of circuits evaluated, with a coefficient of determination R^2 of 0.9903. As the number of circuits to be evaluated increases beyond the number of available cores, the incremental costs of adding new circuits becomes higher; in our observation of execution time for 12 to 256 circuits, our regression analysis provided the equation $T = 247.4k - 410.6$ ms, with $R^2 = 0.998$. This demonstrates that evaluation of large numbers of circuits is optimal when every evaluated circuit can be provided with a dedicated core.

The results above show that as many-way servers are deployed in the cloud, it becomes easier to provide optimal efficiency computing outsourced circuits. A 256-core machine would be able to evaluate 256 circuits in parallel to provide the accepted standard 2^{-80} security parameter. Depending on the computation performed, there can be a trade-off between a slightly weaker security parameter and maintaining *optimal* evaluation on servers with lower degrees of parallelism. In our testbed, optimal evaluation with 12 cores provides a security parameter of $2^{-3.84}$.

	2 Circuits		4 Circuits		8 Circuits	
Program	Outsourced	KSS	Outsourced	KSS	Outsourced	KSS
Millionaires 4	590.0 ± 20%	318.0 ± 5%	749.0 ± 6%	403.0 ± 9%	1100.0 ± 12%	488.0 ± 3%
Millionaires 32	686.0 ± 1%	745.0 ± 2%	1000.0 ± 3%	1030.0 ± 4%	1640.0 ± 2%	1790.0 ± 1%
Millionaires 128	821.0 ± 2%	2091.0 ± 0.5%	1330.0 ± 1%	3207.0 ± 0.8%	2230.0 ± 7%	6128.0 ± 0.9%
Millionaires 1024	1650.0 ± 9%	14710.0 ± 0.3%	2530.0 ± 1%	23580.0 ± 0.8%	4320.0 ± 2%	46180.0 ± 0.6%
Millionaires 8192	6680.0 ± 3%	115200.0 ± 0.2%	10400.0 ± 1%	191000.0 ± 1%	17300.0 ± 1%	388800.0 ± 0.9%
Edit Distance 2	611.0 ± 8%	423.0 ± 4%	880.0 ± 10%	570.0 ± 30%	1200.0 ± 10%	741.0 ± 4%
Edit Distance 4	621.0 ± 6%	519.0 ± 0.7%	858.0 ± 9%	701.0 ± 3%	1300.0 ± 10%	1010.0 ± 3%
Edit Distance 8	669.0 ± 2%	1150.0 ± 2%	934.0 ± 1%	1850.0 ± 3%	1430.0 ± 2%	3230.0 ± 1%
Edit Distance 16	910.0 ± 20%	3460.0 ± 3%	1090.0 ± 3%	6280.0 ± 2%	1670.0 ± 7%	12000.0 ± 2%
Edit Distance 32	1490.0 ± 2%	12140.0 ± 0.7%	1830.0 ± 4%	23420.0 ± 0.5%	2400.0 ± 1%	46400.0 ± 1%
Edit Distance 64	3880.0 ± 1%	47200.0 ± 0.8%	4253.0 ± 0.8%	92370.0 ± 0.9%	5100.0 ± 4%	183600.0 ± 0.5%
Edit Distance 128	15900.0 ± 4%	183800.0 ± 0.1%	16400.0 ± 3%	364100.0 ± 0.7%	17800.0 ± 3%	731200.0 ± 0.5%
Set Intersection 2	684.0 ± 2%	777.0 ± 2%	1020.0 ± 1%	1100.0 ± 10%	1610.0 ± 2%	1850.0 ± 2%
Set Intersection 4	775.0 ± 2%	1459.0 ± 0.7%	1160.0 ± 3%	2350.0 ± 1%	1846.0 ± 0.8%	3750.0 ± 1%
Set Intersection 8	909.0 ± 2%	3370.0 ± 1%	1400.0 ± 10%	5724.0 ± 0.6%	2250.0 ± 5%	9800.0 ± 0.7%
Set Intersection 16	1390.0 ± 1%	9240.0 ± 1%	1880.0 ± 4%	16270.0 ± 0.4%	2760.0 ± 4%	30000.0 ± 1%
Set Intersection 32	3130.0 ± 6%	29000.0 ± 1%	3530.0 ± 2%	54600.0 ± 2%	4560.0 ± 2%	102800.0 ± 0.6%
Set Intersection 64	8570.0 ± 3%	102000.0 ± 1%	9830.0 ± 2%	193600.0 ± 0.2%	10900.0 ± 2%	377800.0 ± 0.5%
Set Intersection 128	31300.0 ± 4%	373600.0 ± 0.2%	32000.0 ± 3%	736900.0 ± 0.9%	33500.0 ± 2%	1467000.0 ± 0.3%
AES-128	1050.0 ± 3%	5636.0 ± 0.9%	1510.0 ± 6%	9400.0 ± 1%	2400.0 ± 2%	15800.0 ± 1%

Table III: Execution time (in ms) of outsourced vs non-outsourced (KSS) evaluation for a subset of circuits. Results with a dash indicate evaluation that the phone was incapable of performing. On 64 core server.

	32 Circuits		128 Circuits		256 Circuits	
Program	Outsourced	KSS	Outsourced	KSS	Outsourced	KSS
Millionaires 4	3060.0 ± 2%	1200.0 ± 2%	15900.0 ± 2%	3870.0 ± 2%	35600.0 ± 1%	7369.0 ± 0.7%
Millionaires 32	5520.0 ± 4%	6180.0 ± 2%	25200.0 ± 1%	23540.0 ± 0.5%	55200.0 ± 0.7%	46450.0 ± 0.6%
Millionaires 128	7770.0 ± 4%	22930.0 ± 0.7%	34200.0 ± 2%	94100.0 ± 1%	73200.0 ± 1%	189400.0 ± 0.9%
Millionaires 1024	16900.0 ± 2%	184000.0 ± 2%	71930.0 ± 0.8%	793600.0 ± 0.9%	147000.0 ± 2%	1612000.0 ± 0.4%
Millionaires 8192	72990.0 ± 0.8 %	1610000.0 ± 0.6 %	330800.0 ± 0.6%	-	728000.0 ± 8%	-
Edit Distance 2	4060.0 ± 5%	2080.0 ± 2%	18100.0 ± 0.8%	7448.0 ± 0.9%	41300.0 ± 1%	14470.0 ± 0.9%
Edit Distance 4	3980.0 ± 5%	3080.0 ± 1%	18200.0 ± 1%	11440.0 ± 0.8%	42000.0 ± 1%	22410.0 ± 0.9%
Edit Distance 8	4920.0 ± 7%	11200.0 ± 1%	21500.0 ± 1%	43200.0 ± 1%	47710.0 ± 0.9%	87100.0 ± 1%
Edit Distance 16	4910.0 ± 3%	44880.0 ± 0.3%	22300.0 ± 3%	182000.0 ± 1%	48700.0 ± 1%	372100.0 ± 0.5%
Edit Distance 32	6620.0 ± 8%	182800.0 ± 0.9%	27200.0 ± 2%	755200.0 ± 0.5%	58250.0 ± 0.6%	-
Edit Distance 64	9960.0 ± 0.5%	740000.0 ± 4%	40400.0 ± 2%	-	85400.0 ± 4%	-
Edit Distance 128	24500.0 ± 3%	2978000.0 ± 0.2%	79500.0 ± 1%	-	159000.0 ± 3%	-
Set Intersection 2	5430.0 ± 3%	6362.0 ± 0.9%	24410.0 ± 0.7%	24160.0 ± 0.7%	54340.0 ± 0.7%	48700.0 ± 1%
Set Intersection 4	6460.0 ± 3%	13890.0 ± 0.7%	28390.0 ± 0.8%	55700.0 ± 0.6%	62960.0 ± 0.5%	115600.0 ± 0.6%
Set Intersection 8	7730.0 ± 2%	34400.0 ± 1%	33800.0 ± 1%	159300.0 ± 0.8%	73310.0 ± 0.8%	343200.0 ± 0.4%
Set Intersection 16	9513.0 ± 0.9%	111600.0 ± 0.5%	41550.0 ± 0.8%	520000.0 ± 1%	88860.0 ± 0.8%	1084000.0 ± 0.5%
Set Intersection 32	13000.0 ± 3%	413800.0 ± 0.7%	55420.0 ± 0.8%	-	116800.0 ± 0.8%	-
Set Intersection 64	19000.0 ± 2%	1596000.0 ± 0.4%	81200.0 ± 1%	-	163600.0 ± 0.7%	-
Set Intersection 128	44500.0 ± 0.8%	6072000.0 ± 0.2%	146500.0 ± 0.5%	-	293900.0 ± 0.7%	-
AES-128	8410.0 ± 2%	58100.0 ± 1%	37300.0 ± 1%	284200.0 ± 0.3%	79400.0 ± 1%	596700.0 ± 0.8%

Table IV: Execution time (in ms) of outsourced vs non-outsourced (KSS) evaluation for a subset of circuits. Results with a dash indicate evaluation that the phone was incapable of performing. On 64 core server.

Clearly more cores would provide stronger security while keeping execution times proportional to our results. A reasonable trade-off might be 32 circuits, as 32-core servers are readily available. Evaluating 32 circuits provides a security parameter of $2^{-10.2}$, equivalent to the adversary having less than a $\frac{1}{512}$ chance of causing the evaluator to compute over a majority of corrupt circuits. Stronger security guarantees on less parallel machines can be achieved at the cost of increasing execution time, as individual cores will not be dedicated to circuit evaluation. However, if a 256-core system is available, it will provide optimal results for achieving a 2^{-80} security parameter.

7.4. Bandwidth

For a mobile device, the costs of transmitting data are intrinsically linked to power consumption, as excess data transmission and reception reduces battery life. Bandwidth is thus a critical resource constraint. In addition, because of potentially uncertain communication channels, transmitting an excess of information can be a rate-limiting factor for circuit evaluation. Figure 7 shows the bandwidth measurement between the phone and remote parties for the edit distance problem with 2 circuits. When we compared execution time for this problem in Figure 3, we found that

	2 Circuits		Factor	4 Circuits		Factor	8 Circuits		Factor
Program	Outsourced	KSS	Improve	Outsourced	KSS	Improve	Outsourced	KSS	Improve
Millionaires 4	2986	4743	1.59X	5403	8476	1.57X	9810	14475	1.48X
Millionaires 32	10322	30291	2.93X	18395	55764	3.03X	31758	95831	3.02X
Millionaires 128	31270	117879	3.77X	54531	217884	4.00X	90194	374747	4.15X
Millionaires 1024	221988	935367	4.21X	382207	1731004	4.53X	616410	2977963	4.83X
Millionaires 8192	1745584	7475271	4.28X	2999319	13835964	4.61X	4817546	23803691	4.94X
Edit Distance 2	4537	9830	2.17X	8205	17970	2.19X	14693	30578	2.08X
Edit Distance 4	4941	34656	7.01X	8893	67350	7.57X	15781	105392	6.68X
Edit Distance 8	6269	186976	29.83X	11309	371446	32.85X	20037	563024	28.10X
Edit Distance 16	7914	840895	106.25X	14119	1678196	118.86X	24504	2526127	103.09X
Edit Distance 32	11736	3490201	297.39X	20803	6974632	335.27X	35566	10476737	294.57X
Edit Distance 64	18880	14150617	749.50X	33171	28291112	852.89X	55690	42463369	762.50X
Edit Distance 128	32684	56910247	1741.23X	56939	113801668	1998.66X	94002	170753027	1816.48X
Set Intersection 2	10090	42881	4.25X	17990	81079	4.51X	31092	133429	4.29X
Set Intersection 4	17236	144507	8.38X	30360	279977	9.22X	51218	443687	8.66X
Set Intersection 8	31048	505129	16.27X	54140	992513	18.33X	89550	1536313	17.16X
Set Intersection 16	58736	1851451	31.52X	101828	3667741	36.02X	166470	5596799	33.62X
Set Intersection 32	113216	7059111	62.35X	195412	14048229	71.89X	316726	21262819	67.13X
Set Intersection 64	221920	27648643	124.59X	382068	55157629	144.37X	616214	83117495	134.88X
Set Intersection 128	439328	109658303	249.60X	755380	219037621	289.97X	1215190	329318635	271.00X
AES-128	33120	821303	24.80X	58276	1614121	27.70X	97814	2512507	25.69X

Table V: Total Bandwidth (Bytes) transmitted to and from the phone during execution.

	16 Circuits		Factor	32 Circuits		Factor	256 Circuits		Factor
Program	Outsourced	KSS	Improve	Outsourced	KSS	Improve	Outsourced	KSS	Improve
Millionaires 4	19051	27941	1.47X	37533	54873	1.46X	298843	440723	1.47X
Millionaires 32	61267	186845	3.05X	120285	368873	3.07X	963235	2982539	3.10X
Millionaires 128	172379	731621	4.24X	336749	1445369	4.29X	2703083	11696723	4.33X
Millionaires 1024	1171051	5816197	4.97X	2280333	11492665	5.04X	18327691	93029107	5.08X
Millionaires 8192	9143243	46492805	5.08X	17794637	91871033	5.16X	143049611	-	-
Edit Distance 2	28517	59467	2.09X	56165	117245	2.09X	448325	948169	2.11X
Edit Distance 4	30573	208823	6.83X	60157	415685	6.91X	480429	3475829	7.23X
Edit Distance 8	38845	1123543	28.92X	76461	2244581	29.36X	611197	19003285	31.09X
Edit Distance 16	47299	5048661	106.74X	92889	10093729	108.66X	743299	85684707	115.28X
Edit Distance 32	68463	20947705	305.97X	134257	41889641	312.01X	1075599	-	-
Edit Distance 64	106791	84916617	795.17X	208993	169823113	812.58X	1676199	-	-
Edit Distance 128	179575	341487229	1901.64X	350721	682955633	1947.29X	2815447	-	-
Set Intersection 2	59994	262176	4.37X	117798	519670	4.41X	943242	4268862	4.53X
Set Intersection 4	98324	878338	8.93X	192536	1747640	9.08X	1543844	14561248	9.43X
Set Intersection 8	171144	3054882	17.85X	334332	6092020	18.22X	2683608	51197760	19.08X
Set Intersection 16	317296	11158438	35.17X	618948	22281716	36.00X	4971328	188228740	37.86X
Set Intersection 32	602432	42455646	70.47X	1173844	84841300	72.28X	9432080	-	-
Set Intersection 64	1170656	166095334	141.88X	2279540	332051012	145.67X	18320816	-	-
Set Intersection 128	2307104	658358286	285.36X	4490932	1316437588	293.13X	36098288	-	-
AES-128	187664	4996530	26.62X	367364	9964576	27.12X	2947808	83640720	28.37X

Table VI: Total Bandwidth (Bytes) transmitted to and from the phone during execution.

	32 Circuits Time (ms)	64 Circuits (ms)	128 Circuits (ms)	Optimized Gates	Unoptimized Gates	Size (MB)
RSA128	505000.0 $\pm$ 2%	734000.0 $\pm$ 4%	1420000.0 $\pm$ 1%	116,083,727	192,537,834	774
Dijkstra20	25800.0 $\pm$ 2%	49400.0 $\pm$ 1%	106000.0 $\pm$ 1%	1,653,542	20,288,444	11
Dijkstra50	135000.0 $\pm$ 1%	197000.0 $\pm$ 3%	389000.0 $\pm$ 2%	22,109,732	301,846,263	147
Dijkstra100	892000.0 $\pm$ 2%	1300000.0 $\pm$ 2%	2560000.0 $\pm$ 1%	168,422,382	2,376,377,302	1124

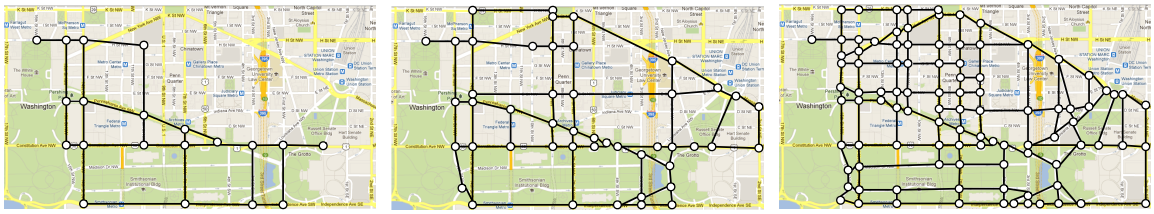
Table VII: Execution time for evaluating a 128-bit blinded RSA circuit and Dijkstra shortest path solvers over graphs with 20, 50, and 100 vertices. All numbers are for outsourced evaluation, as the circuits are too large to be computed without outsourcing to a proxy.

	32 Circuits	64 Circuits	128 Circuits
RSA128	334629	672067	1346943
Dijkstra20	3862280	7770598	15587234
Dijkstra50	9575622	19266732	38648952
Dijkstra100	19087192	38405622	77042482

Table VIII: Bandwidth of 128-bit RSA and Dijkstra 20, 50, and 100. All entries are in Bytes.

	16 Circuits			64 Circuits			256 Circuits		
Latency	0 ms	100 ms	500 ms	0 ms	100 ms	500 ms	0 ms	100 ms	500 ms
Millionaires 128	3950.0 ± 1%	13600.0 ± 7%	41900.0 ± 2%	15000.0 ± 2%	37400.0 ± 2%	143000.0 ± 2%	73200.0 ± 1%	157000.0 ± 1%	528800.0 ± 0.5%
Millionaires 1024	8484.0 ± 0.7%	20400.0 ± 3%	53500.0 ± 2%	34800.0 ± 1%	63500.0 ± 1%	197500.0 ± 0.7%	147000.0 ± 2%	259000.0 ± 3%	758000.0 ± 5%
Millionaires 8192	36000.0 ± 1 %	67700.0 ± 1%	133000.0 ± 1%	191000.0 ± 5%	235000.0 ± 6%	529000.0 ± 5%	728000.0 ± 8%	1100000.0 ± 11%	2300000.0 ± 11%
Edit Distance 16	2680.0 ± 4%	10700.0 ± 3%	43200.0 ± 2%	9490.0 ± 2%	32600.0 ± 2%	138000.0 ± 1%	48700.0 ± 1%	131500.0 ± 0.4%	508400.0 ± 0.6%
Edit Distance 32	3740.0 ± 6 %	12200.0 ± 3%	48400.0 ± 2%	12100.0 ± 1 %	35300.0 ± 2%	146000.0 ± 2%	58250.0 ± 0.7 %	141000.0 ± 1%	523400.0 ± 0.5%
Edit Distance 128	20500.0 ± 3%	25800.0 ± 2%	64800.0 ± 2%	38800.0 ± 1%	56620.0 ± 0.8%	183000.0 ± 2%	159000.0 ± 3%	232500.0 ± 0.8%	654000.0 ± 1%
Set Intersection 16	4940.0 ± 1%	13700.0 ± 5%	48100.0 ± 2%	23000.0 ± 4%	41400.0 ± 2%	146400.0 ± 0.4%	88860.0 ± 0.8%	175600.0 ± 0.6%	583100.0 ± 0.9%
Set Intersection 32	6740.0 ± 2 %	16400.0 ± 6%	50600.0 ± 2%	33200.0 ± 8%	48400.0 ± 2%	155000.0 ± 3%	116800.0 ± 0.8%	207000.0 ± 1%	627300.0 ± 0.8%
Set Intersection 128	38600.0 ± 2%	44100.0 ± 1%	108000.0 ± 2%	72700.0 ± 1%	105000.0 ± 2%	284000.0 ± 2%	293900.0 ± 0.7%	432000.0 ± 4%	1040000.0 ± 4%
AES	4390.0 ± 1%	13300.0 ± 5%	46400.0 ± 2%	17100.0 ± 2%	40000.0 ± 1%	144000.0 ± 1%	79400.0 ± 1%	159700.0 ± 0.6%	535400.0 ± 0.7%

Table IX: Network Latency Results.



(a) 20 identified intersections. (b) 50 identified intersections. (c) 100 identified intersections.

Fig. 8: Map of potential presidential motorcade routes through Washington, DC. As the circuit size increases, a larger area can be represented at a finer granularity.

trivially small circuits could execute in less time without outsourcing. Note, however, that *there are no cases where the non-outsourced scheme consumes less bandwidth than with outsourcing*.

This is a result of the significant improvements garnered by using our outsourced oblivious transfer (OOT) construction described in Section 4. Recall that with the OOT protocol, the mobile device sends inputs for evaluation to the generator; however, after this occurs, all further evaluation until the final output verification from the cloud proxy occurs between the generator and the proxy, ensuring that further communication is not required by the mobile device. Figure 7 shows that the amount of data transferred increases only nominally compared to the non-outsourced protocol. Apart from the initial set of inputs transmitted to the generator, data demands are largely constant. This is further reflected in Tables V and VI, which shows the vast bandwidth savings over the 32-circuit evaluation of our representative programs. In particular, for large, complex circuits, the savings are vast: outsourced AES-128 requires 96.3% less bandwidth, while set intersection of size 128 requires 99.7% less bandwidth than in the non-outsourced evaluation. Remarkably, the edit distance 128 problem requires 99.95%, *over 1900 times less bandwidth*, for outsourced execution.

7.5. Network Latency

As network latency is a limiting factor on mobile phones, we wanted to see how our outsourced system performed in an environment with latency. We performed trials of our execution system with two different amounts of latency added, 100ms and 500ms. It was found by Huang [**?**] that the median ping latency to a landserver on a 3G connection was between 180ms to 250ms.

In Table IX we present the results of our latency tests. The slowdown of added latency is not uniform across all of our tests. With the *Millionaires 8192*, we observed a slowdown of about 1.9X to 1.2X from 0 latency to 100ms latency, depending on the amount of circuits executed. Whereas *Set Intersection 16* had a slowdown of 2.8X to 2X when we added 100ms of latency. Making the transition from 0 latency to 500 ms makes the differences more apparent. We observed a slowdown of 3.7X to 2.8X for *Millionaires 8192*. Correspondingly, the *Set Intersection 16* had an observed slowdown between 9.7X to 6.4X.

The reason for the difference in the slowdown is due to the different bottlenecks different programs have in our system. For some programs the bottleneck will be at the phases necessary for input for the different parties, the oblivious transfer (large mobile input) and consistency check (large generator input). For other programs the bottleneck will be the garbled circuit generation and evaluation. Our goal is to improve the performance of our system in high latency environments in the future.

8. EVALUATING LARGE CIRCUITS

Beyond the standard benchmarks for comparing garbled circuit execution schemes, we aimed to provide compelling applications that exploit the mobile platform with large circuits that would be used in real-world scenarios. We discuss public-key cryptography and the Dijkstra shortest path algorithm, then describe how the latter can be used to implement a privacy-preserving navigation application for mobile phones.

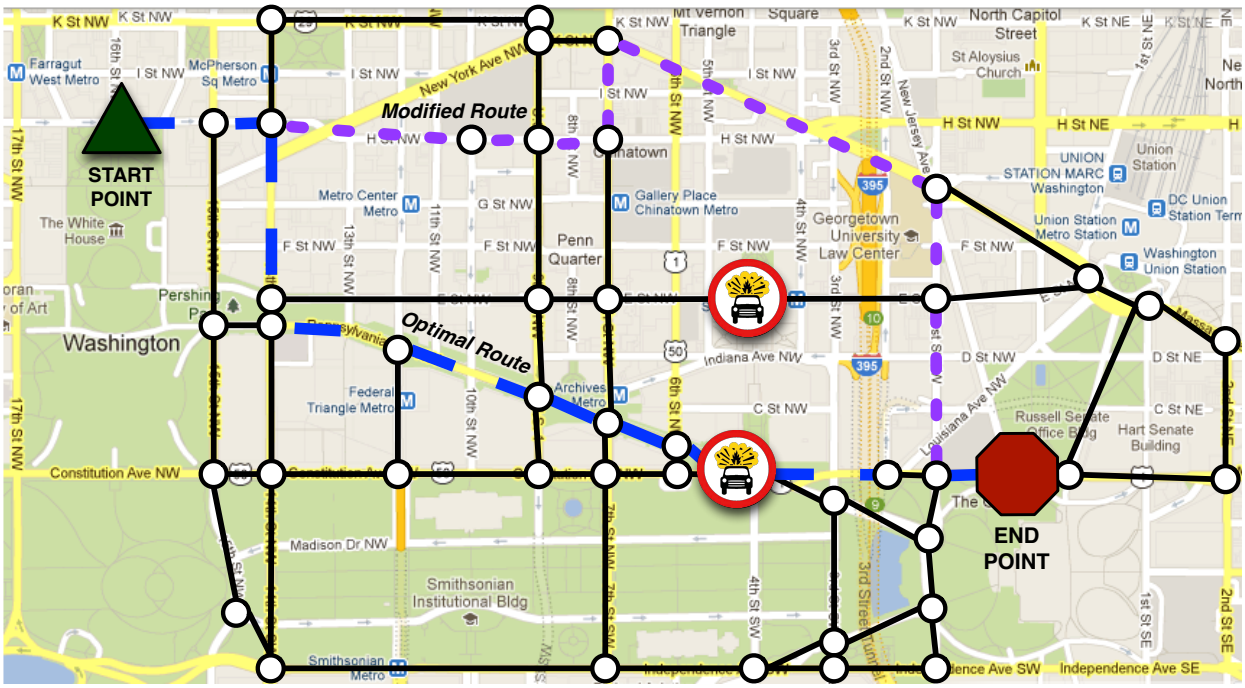


Fig. 9: Motorcade route with hazards along the route. The dashed blue line represents the optimal route, while the dotted violet line represents the modified route that takes hazards into account.

8.1. Large Circuit Benchmarks

Table VII shows the execution time required for a blinded RSA circuit of input size 128. For these tests we used our more powerful 64 core server. Our testbed is able to give dedicated CPUs when running 32 circuits in parallel. Each circuit would have 1 core for the generation and 1 core for the evaluation. As described in Section 7, larger testbeds capable of executing 128 or 256 cores in parallel would be able to provide similar results for executing the 256 circuits necessary for a 2^{-80} security parameter as they could evaluate the added circuits in parallel. The main difference in execution time would come from the multiple OTs from the mobile device to the outsourced proxy. The RSA circuit has been previously evaluated with *KSS*, but never from the standpoint of a mobile device.

We only report the outsourced execution results, as the circuits are far too large to evaluate directly on the phone. As with the larger circuits described in Section 7, the phone runs out of memory from merely trying to store a representation of the circuit. Prior to optimization, the blinded RSA circuit is $192,537,834$ gates and afterward, comprises $116,083,727$ gates, or 774 MB in size.

The implementation of Dijkstra's shortest-path algorithm results in very large circuits. As shown in Table VII, the pre-optimized size of the shortest path circuit for 20 vertices is $20,288,444$ gates and after optimization is $1,653,542$ gates. The 100-node graph is even larger, with $168,422,382$ gates post optimization, 1124 MB in size. This final example is among the largest publicly evaluated circuit to date. While it may be possible for existing protocols to evaluate circuits of similar size, it is significant that we are evaluating comparably massive circuits from a resource-constrained mobile device. Table VIII gives the amount of bandwidth these larger programs use.

8.2. Privacy-Preserving Navigation

Mapping and navigation are some of the most popular uses of a smartphone. Consider how directions may be given using a mobile device and an application such as Google Maps, without revealing the user's current location, their ultimate destination, or the route that they are following. That is, the navigation server should remain oblivious of these details to ensure their mutual privacy and to prevent giving away potentially sensitive details if the phone is compromised. Specifically, consider planning of the motorcade route for the recent Presidential inauguration. In this case, the route is generally known in advance but is potentially subject to change if sudden threats emerge. A field agent along the route wants to receive directions without providing the navigation service any additional details, and without sensitive information about the route loaded to the phone. Moreover, because the threats may be classified, the navigation service does not want the holder of the phone to be given this information directly.

To model this scenario, we overlay a graph topology on a map of downtown Washington D.C., encoding intersections as vertices. Edge weights are a function of their distance and heuristics such as potential risks along a graph

edge. Figure 8 shows graphs generated based on vertices of 20, 50, and 100 nodes, respectively. Note that the 100-node graph (Figure 8c) encompasses a larger area and provides finer-grained resolution of individual intersections than the 20-node graph (Figure 8a).

There is a trade-off between detail and execution time, however; as shown in Table VII, a 20-vertex graph can be evaluated in under 26 seconds, while a 100-vertex graph requires almost 15 minutes with 32 circuits in our 64-core server testbed. The 64 circuit evaluation requires more time: almost 50 seconds for the 20-vertex graph, and almost 22 minutes for a 100-vertex graph. We anticipate that based on the role a particular agent might have on a route, they will be able to generate a route that covers their particular geographical jurisdiction and thus have an appropriately-sized route, with only certain users requiring the highest-resolution output. Additionally, as described in Section 7.3, servers with more parallel cores can simultaneously evaluate more circuits, giving faster results for the 64 circuit evaluation.

Figure 9 reflects two routes. The first, overlaid with a dashed blue line, is the shortest path under optimal conditions that is output by our directions service, based on origin and destination points close to the historical start and end points of the past six presidential inaugural motorcades. Now consider that incidents have happened along the route, shown in the figure as a car icon in a hazard zone inside a red circle. The agent recalculates the optimal route, which has been updated by the navigation service to assign severe penalties to those corresponding graph edges. The updated route returned by the navigation service is shown in the figure as a path with a dotted purple line. In the 50-vertex graph in Figure 8, the updated directions would be available in just over 135 seconds for 32-circuit evaluation, and 196 and a half seconds for 64-circuit evaluation.

9. CONCLUSION

While garbled circuits offer a powerful tool for secure function evaluation, they typically assume participants with massive computing resources. Our work solves this problem by presenting a protocol for outsourcing garbled circuit evaluation from a resource-constrained mobile device to a cloud provider in the malicious setting. By extending existing garbled circuit evaluation techniques, our protocol significantly reduces both computational and network overhead on the mobile device while still maintaining the necessary checks for malicious or lazy behavior from all parties. Our outsourced oblivious transfer construction significantly reduces the communication load on the mobile device and can easily accommodate more efficient OT primitives as they are developed. The performance evaluation of our protocol shows dramatic decreases in required computation and bandwidth. For the edit distance problem of size 128 with 32 circuits, computation is reduced by 98.92% and bandwidth overhead reduced by 99.95% compared to non-outsourced execution. These savings are illustrated in our privacy-preserving navigation application, which allows a mobile device to efficiently evaluate a massive garbled circuit securely through outsourcing. These results demonstrate that the recent improvements in garbled circuit efficiency can be applied in practical privacy-preserving mobile applications on even the most resource-constrained devices.

ACKNOWLEDGMENTS

This material is based on research sponsored by DARPA under agreement number FA8750-11-2-0211. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. We would like to thank Adam Bates for his help with figures for this paper; Daniel Ellsworth for his help with setting up our testbed; Benjamin Kreuter, abhi shelat, and Chih-hao Shen for working with us on their garbled circuit compiler and evaluation framework; Chris Peikert for providing helpful feedback on our proofs of security; Thomas DuBuisson and Galois for their assistance in the performance evaluation; and Ian Goldberg for his guidance in shepherding the conference version of this work.

REFERENCES

Mihir Bellare and Silvio Micali. 1990. Non-Interactive Oblivious Transfer and Applications. In *Advances in Cryptology–CRYPTO*.

Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the annual ACM symposium on Theory of computing*.

Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic encryption and multiparty computation. In *Proceedings of the Annual international conference on Theory and applications of cryptographic techniques*.

Justin Brickell and Vitaly Shmatikov. 2005. Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of the international conference on Theory and Application of Cryptology and Information Security*.

Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. 2002. Universally composable two-party and multi-party secure computation. In *Proceedings of the annual ACM symposium on Theory of computing*.

Henry Carter, Chaitrali Amrutkar, Italo Dacosta, and Patrick Traynor. To appear 2013. For your phone only: custom protocols for efficient secure function evaluation on mobile devices. *Journal of Security and Communication Networks (SCN)* (To appear 2013).

Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin Butler. 2012. *Secure Outsourced Garbled Circuit Evaluation for Mobile Devices*. Technical Report GT-CS-12-09. College of Computing, Georgia Institute of Technology.

David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the annual ACM symposium on Theory of computing*.

Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. 2012. On the security of the "Free-XOR" technique. In *Proceedings of the international conference on Theory of Cryptography*.

Ivan Damgård and Yuval Ishai. 2006. Scalable secure multiparty computation. In *Proceedings of the annual international conference on Advances in Cryptology*.

Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and unconditionally secure multiparty computation. In *Proceedings of the annual international cryptology conference on Advances in cryptology*.

Ivan Damgard, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*.

Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic Evaluation of the AES Circuit. In *Advances in Cryptology - CRYPTO*.

Oded Goldreich and Ariel Kahan. 1995. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology* 9 (1995), 167–189.

Vipul Goyal, Payman Mohassel, and Adam Smith. 2008. Efficient two party and multi party computation against covert adversaries. In *Proceedings of the theory and applications of cryptographic techniques annual international conference on Advances in cryptology*.

Matthew Green, Susan Hohenberger, and Brent Waters. 2011. Outsourcing the Decryption of ABE Ciphertexts. In *Proceedings of the USENIX Security Symposium*.

Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. 2010. TASTY: tool for automating secure two-party computations. In *Proceedings of the ACM conference on Computer and Communications Security*.

Yan Huang, Peter Chapman, and David Evans. 2011. Privacy-Preserving Applications on Smartphones. In *Proceedings of the USENIX Workshop on Hot Topics in Security*.

Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS '12: Proceedings of the 19th ISOC Symposium on Network and Distributed Systems Security*. San Diego, CA, USA.

Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure Two-Party Computation Using Garbled Circuits. In *Proceedings of the USENIX Security Symposium*.

Yan Huang, Jonathan Katz, and David Evans. 2012. Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Alexander Iliev and Sean W. Smith. 2010. Small, Stupid, and Scalable: Secure Computing with Faerieplay. In *The ACM Workshop on Scalable Trusted Computing*.

Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *Proceedings of the Annual International Cryptology Conference*.

Somesh Jha, Louis Kruger, and Vitaly Shmatikov. 2008. Towards Practical Privacy for Genomic Computation. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Seny Kamara, Payman Mohassel, and Mariana Raykova. 2011. Outsourcing Multi-Party Computation. Cryptology ePrint Archive, Report 2011/272. (2011). http://eprint.iacr.org/.

Seny Kamara, Payman Mohassel, and Ben Riva. 2012. Salus: A System for Server-Aided Secure Function Evaluation. In *Proceedings of the ACM conference on Computer and communications security (CCS)*.

Mehmet S. Kiraz. 2008. *Secure and Fair Two-Party Computation*. Ph.D. Dissertation. Technische Universiteit Eindhoven.

Mehmet S. Kiraz and Berry Schoenmakers. 2006. A protocol issue for the malicious case of Yaos garbled circuit construction. In *Proceedings of Symposium on Information Theory in the Benelux*.

Vladimir Kolesnikov and Thomas Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *Proceedings of the international colloquium on Automata, Languages and Programming, Part II*.

Benjamin Kreuter, Benjamin Mood, abhi shelat, and Kevin Butler. 2013. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In *Proceedings of the USENIX Security Symposium*.

Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. 2012. Billion-Gate Secure Computation with Malicious Adversaries. In *Proceedings of the USENIX Security Symposium*.

Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh. 2006. Secure Function Evaluation with Ordered Binary Decision Diagrams. In *Proceedings of the ACM conference on Computer and communications security (CCS)*.

Yehuda Lindell. 2008. Lower Bounds and Impossibility Results for Concurrent Self Composition. *Journal of Cryptology* 21, 2 (2008), 200–249.

Yehuda Lindell and Benny Pinkas. 2000. Privacy preserving data mining. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology*.

Yehuda Lindell and Benny Pinkas. 2007. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *Proceedings of the annual international conference on Advances in Cryptology*.

Yehuda Lindell and Benny Pinkas. 2011. Secure two-party computation via cut-and-choose oblivious transfer. In *Proceedings of the conference on Theory of cryptography*.

Lior Malka. 2011. VMCrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security*.

Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. 2004. Fairplay–a secure two-party computation system. In *Proceedings of the USENIX Security Symposium*.

Payman Mohassel and Matthew Franklin. 2006. Efficiency Tradeoffs for Malicious Two-Party Computation. In *Proceedings of the Public Key Cryptography conference*.

Benjamin Mood, Lara Letaw, and Kevin Butler. 2012. Memory-Efficient Garbled Circuit Generation for Mobile Devices. In *Proceedings of the IFCA International Conference on Financial Cryptography and Data Security (FC)*.

Moni Naor and Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the annual ACM symposium on Theory of computing*.

Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *Proceedings of the annual ACM-SIAM symposium on Discrete algorithms*.

Moni Naor, Benny Pinkas, and Reuben Sumner. 1999. Privacy preserving auctions and mechanism design. In *Proceedings of the ACM conference on Electronic commerce*.

Nilesh Nipane, Italo Dacosta, and Patrick Traynor. 2011. "Mix-In-Place" anonymous networking using secure function evaluation. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. 2010. SCiFI A System for Secure Face Identification. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer. In *Advances in Cryptology (CRYPTO)*.

Wayne Rash. 2012. Dropbox Password Breach Highlights Cloud Security Weaknesses. http://www.eweek.com/c/a/Security/Dropbox-Password-Breach-Highlights-Cloud-Security-Weaknesses-266215/. (2012).

abhi shelat and Chih-Hao Shen. 2011. Two-output secure computation with malicious adversaries. In *Proceedings of the Annual international conference on Theory and applications of cryptographic techniques*.

Keir Thomas. 2010. Microsoft Cloud Data Breach Heralds Things to Come. http://www.pcworld.com/article/214775/microsoft_cloud_data_breach_sign_of_future.html. (2010).

Andrew C. Yao. 1982. Protocols for secure computations. In *Proceedings of the Annual Symposium on Foundations of Computer Science*.